5

Elements of differential geometry

This chapter presents various concepts useful to fully understand curves and surfaces discussed in later sections. In particular, we introduce here the notion of a curve as a point-valued function of a real variable, the variable-free representation of functions and the Fréchet derivative for functions of one and several variables. This derivative gives the framework for introducing the concepts of manifold, tangent space and vector field from a geometrical perspective. Then, more standard topics on curves are briefly outlined, including tangent, curvature, centre of curvature and osculating circles. Finally, some elements of differential geometry of surfaces are discussed, including first and second fundamental forms and Gauss curvature. As usual in this book, the mathematics is normally given without proofs, but is often coupled with a `PLaSM` implementation of the important ideas. We hope that the implementation and the examples may help to clarify subtle mathematical details.

5.1 Curves

A *curve* in $\mathbb{E}^d$ is a point-valued mapping defined by summing to the origin of a Cartesian system $\{\boldsymbol{o}, (\boldsymbol{e}_i)\}$ a vector-valued function $\boldsymbol{\alpha} : \mathbb{R} \to \mathbb{R}^d$ of a real parameter, so that a point of the curve is generated as:

$$\boldsymbol{c}(u) = \boldsymbol{o} + \boldsymbol{\alpha}(u), \qquad u \in [a, b] \subset \mathbb{R}.$$

The *image* of the curve is the set $\boldsymbol{c}[a, b]$ of its $\mathbb{E}^d$ points. The *domain* of the curve is the parameter interval $[a, b]$, i.e. the set $\{u \in \mathbb{R} | a \leq u \leq b\}$, often normalized to the standard unit interval $[0, 1]$.

The important part of the curve definition is the vector-valued function $\boldsymbol{\alpha}$, so that sometimes we use the word "curve" for it. From a notational viewpoint, we normally use a bold Latin letter, say $\boldsymbol{a}$, $\boldsymbol{b}$ or $\boldsymbol{c}$, to indicate a map $\mathbb{R} \to \mathbb{E}^d$, and a bold Greek letter, say $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ or $\boldsymbol{\gamma}$, to indicate a map $\mathbb{R} \to \mathbb{R}^d$. This convention will sometimes be broken according to the standard usage of symbols, e.g. for the *intrinsic triplet* of vector functions $\mathbb{R} \to \mathbb{R}^3$, that are denoted by $\boldsymbol{t}$, $\boldsymbol{n}$ and $\boldsymbol{b}$, respectively.

Example 5.1.1 (3D curve)
A curve $\boldsymbol{c}$ in three-dimensional space has 3 *coordinate functions*, and is often denoted

Geometric Programming for Computer-Aided Design Alberto Paoluzzi
 ISBN 0-471-89942-9

as

$$\boldsymbol{c}(u) = (x(u), y(u), z(u))^T, \qquad u \in [a, b]$$

where $x(u) = \boldsymbol{\alpha}(u) \cdot \boldsymbol{e}_1$, $y(u) = \boldsymbol{\alpha}(u) \cdot \boldsymbol{e}_2$ and $z(u) = \boldsymbol{\alpha}(u) \cdot \boldsymbol{e}_3$. The *tangent* vector function $\boldsymbol{t}$ is defined by

$$\boldsymbol{t}(u) = \frac{d}{du}\boldsymbol{c}(u) = \frac{d}{du}\boldsymbol{\alpha}(u) = (x'(u), y'(u), z'(u))^T, \qquad u \in [a, b].$$

5.1.1 Variable-free notation

Accordingly with the functional approach of the `PLaSM` language, we often denote a 3D curve, as well as its derivative curves, by using the variable-free notation:

$$\boldsymbol{c} = (x, y, z)^T$$

with $x = \boldsymbol{\alpha} \cdot \boldsymbol{e}_1$, $y = \boldsymbol{\alpha} \cdot \boldsymbol{e}_2$, and $z = \boldsymbol{\alpha} \cdot \boldsymbol{e}_3$.

It should be clearly understood that x, y, z are here maps $\mathbb{R} \to \mathbb{R}$ and that each $\boldsymbol{e}_i$ has the constant maps $\underline{0} : \mathbb{R} \to 0$ and $\underline{1} : \mathbb{R} \to 1$ as components.

Analogously, we write a curve $\boldsymbol{c} : \mathbb{R} \to \mathbb{E}^d$, as $\boldsymbol{c} = \boldsymbol{o} + \boldsymbol{\alpha}$, with $\boldsymbol{\alpha} = (\alpha_i)$, where $\alpha_i : \mathbb{R} \to \mathbb{R}$, for all i. The variable-free notation, where functions are directly added and multiplied, exactly like numbers, is very useful for quickly and easily implementing curves and surfaces in our language.

Useful maps Some special maps are needed to perform such variable-free calculus with functions. As the reader already knows, they have a direct translation in `PLaSM`.

1. $\mathrm{id} : \mathbb{R} \to \mathbb{R}; x \mapsto x$ (identity)
2. $\underline{c} : \mathbb{R} \to \mathbb{R}; x \mapsto c$ (constant)
3. $\sigma : \{1, \ldots, d\} \times \mathbb{R}^d \to \mathbb{R}; (i, (x_1, \ldots, x_d)) \mapsto x_i$ (selection)

A computer scientist would probably prefer the following specification, just to point out that σ is often used as a *partial* function, i.e. a function which may be applied to a subset of its arguments:

3. $\sigma : \{1, \ldots, d\} \to (\mathbb{R}^d \to \mathbb{R}); i \mapsto ((x_1, \ldots, x_d) \mapsto x_i)$ (selection)

Actually, the `FL` primitives `ID`, `K` and `SEL` used by the `PLaSM` language have no domain restrictions, and can be applied to any type of data objects.

Algebraic operations We also need to recall how to perform algebraic operations in the linear algebra of maps $\mathbb{R} \to \mathbb{R}$. For each map $\boldsymbol{\alpha}, \boldsymbol{\beta} : \mathbb{R} \to \mathbb{R}$ and each scalar $a \in \mathbb{R}$

$$\boldsymbol{\alpha} + \boldsymbol{\beta} : u \mapsto \boldsymbol{\alpha}(u) + \boldsymbol{\beta}(u), \quad \boldsymbol{\alpha}\boldsymbol{\beta} : u \mapsto \boldsymbol{\alpha}(u)\boldsymbol{\beta}(u), \quad a\,\boldsymbol{\beta} : u \mapsto a\,\boldsymbol{\beta}(u).$$

Consequently we have that

$$\boldsymbol{\alpha} - \boldsymbol{\beta} : u \mapsto \boldsymbol{\alpha}(u) - \boldsymbol{\beta}(u), \quad \boldsymbol{\alpha}/\boldsymbol{\beta} : u \mapsto \boldsymbol{\alpha}(u)/\boldsymbol{\beta}(u).$$

Coordinate representation Finally, remember that the coordinate functions of a curve $\boldsymbol{\alpha} = (\alpha_i)$ are maps $\mathbb{R} \to \mathbb{R}$. The variable-free vector notation stands for the linear combination of coordinate functions with the basis vectors of the target space:

$$(\alpha_1, \cdots, \alpha_d)^T : \mathbb{R} \to \mathbb{R}^d;\ u \mapsto \sum_{i=1}^{d} \alpha_i \boldsymbol{e}_i.$$

Example 5.1.2 (Circular arc)
Some different curves are given here. They have the same image in $\mathbb{E}^2$ but different coordinate representation in the space of functions $\mathbb{R} \to \mathbb{R}$. All such curves generate a circular arc of unit radius centered at the origin.

1. trigonometric representation:

$$\boldsymbol{\alpha}(u) = \left(\cos\left(\frac{\pi}{2}u\right), \sin\left(\frac{\pi}{2}u\right)\right)^T \qquad u \in [0,1]$$

2. rational representation:

$$\boldsymbol{\beta}(u) = \left(\frac{1-u^2}{1+u^2}, \frac{2u}{1+u^2}\right)^T \qquad u \in [0,1]$$

3. Cartesian representation:

$$\boldsymbol{\gamma}(u) = \left(u, \sqrt{1-u^2}\right)^T \qquad u \in [0,1]$$

It is possible to verify that the image sets of such curves coincide, i.e. that $\boldsymbol{\alpha}[0,1] = \boldsymbol{\beta}[0,1] = \boldsymbol{\gamma}[0,1]$.

Example 5.1.3 (Variable-free circular arc)
It may be useful to give the variable-free representation of the three maps on the $[0,1]$ interval shown in Example 5.1.2, that is exactly the representation we need to give a `PLaSM` implementation of such maps, provided in Script 5.1.2:

$$\boldsymbol{\alpha} = \left(\cos \circ \left(\frac{\pi}{\underline{2}}\mathrm{id}\right),\ \sin \circ \left(\frac{\pi}{\underline{2}}\mathrm{id}\right)\right)^T \tag{5.1}$$

$$\boldsymbol{\beta} = \left(\frac{\underline{1} - \mathrm{id}^2}{\underline{1} + \mathrm{id}^2},\ \frac{\underline{2}\,\mathrm{id}}{\underline{1} + \mathrm{id}^2}\right)^T \tag{5.2}$$

$$\boldsymbol{\gamma} = \left(\mathrm{id},\ \mathrm{id}^{\frac{1}{2}} \circ (\underline{1} - \mathrm{id}^2)\right)^T \tag{5.3}$$

Script 5.1.1 (Toolbox)

```
DEF interval (a,b::IsReal)(n::IsIntPos) = (T:1:a ~ QUOTE ~ #:n):((b-a)/n);
DEF interval2D (a1,a2,b1,b2::IsReal)(n1,n2::IsIntPos) =
   interval:<a1,b1>:n1 * interval:<a2,b2>:n2;
```

Toolbox Some predicates and functions needed by the operators in this chapter are given in Script 5.1.1. In particular, the `interval` operator provides a simplicial decomposition with `n` elements of the real interval $[\mathtt{a}, \mathtt{b}]$, whereas the `interval2D` operator returns a decomposition with `n1` $\times$ `n2` subintervals of the domain $[\mathtt{a1}, \mathtt{b1}] \times [\mathtt{a2}, \mathtt{b2}] \subset \mathbb{R}^2$.

Few other functions of general utility are used in the remainder of this chapter. In particular, the `SQR` function, that returns its squared input, was defined in Script 3.2.4; the predicates `IsVect` and `IsPoint` were given in Script 3.1.1 and 3.2.3, respectively. The vector operations used here were defined in Chapter 3.

Implementation The circle segment representations of Example 5.1.3 are directly used in the `PLaSM` implementation of curves in Script 5.1.2. To understand the implementation, notice that we generate a polyhedral complex by mapping the curve vector function (either $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ or $\boldsymbol{\gamma}$ of Example 5.1.3) on the polyhedral representation of the $[0, 1] \subset \mathbb{R}$ domain.

According to the semantics of the `MAP` function, the curve mapping is applied to all vertices of a simplicial decomposition of the polyhedral domain. But all vertices are represented as *sequences* of coordinates, say <*u*> for a curve, so that in order to act on u the mapping must necessarily *select* it from the sequence.

Hence we might substitute each *id* function instance with the `PLaSM` denotation `S1` for the $\sigma(1)$ selector function.

Exactly the same result is obtained by using either $\boldsymbol{\alpha} \circ \sigma(1)$, $\boldsymbol{\beta} \circ \sigma(1)$ or $\boldsymbol{\gamma} \circ \sigma(1)$, as done in the following code.

Script 5.1.2 (Circular arc maps)

```
DEF SQRT  = ID ** K:(1/2);

DEF alpha = < cos ~ (K:(PI/2) * ID), sin ~ (K:(PI/2) * ID) >;
DEF beta  = < (K:1 - SQR)/(K:1 + SQR), (K:2 * ID)/(K:1 + SQR) >;
DEF gamma = < ID, SQRT ~ (K:1 - SQR) >;

MAP:(CONS:alpha ~ S1):(interval:<0,1>:10);
MAP:(CONS:beta  ~ S1):(interval:<0,1>:10);
MAP:(CONS:gamma ~ S1):(interval:<0,1>:10);
```

Remarks Let us note that, e.g., `alpha` is a *sequence* of coordinate functions. Conversely, `CONS:alpha` is the correct implementation of the *vector-valued* function $\boldsymbol{\alpha}$, which only can be *composed* with other functions, say `S1`.

Notice also that `SQR` (square), given in Script 3.2.4, is the `PLaSM` implementation of the id^2 function and that the language explicitly requires the operator `*` to denote the

product of functions.

Finally, we remark that `SQRT` (square root), which is actually primitive in `PLaSM`, can be also defined easily using standard algebraic rules, where `**` is the predefined power operator.

Example 5.1.4 (Comparing parametrizations)
The parametrizations of the arcs generated by maps $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ are quite different. It is possible to see this fact by looking at Figure 5.1, where the graphical markers associated with the mapped image of each point of a uniform sampling of the curve domain generated by Script 5.1.3 are shown. The `polymarker` and `markerSize` implementation is given in Script 7.2.9. Notice that the functional expression

```
polymarker:1 ~ S1 ~ UKPOL
```

converts its polyhedral input into a triplet of vertices, cells and polyhedra, then extracts the vertices, and finally gives them as arguments to the function `polymarker:1`, which attaches a marker of type `1` to each point.

Script 5.1.3 (Curve sampling marking)

```
DEF markerSize = 0.05;
DEF markers = polymarker:1 ~ S1 ~ UKPOL;

(markers ~ MAP:(CONS:alpha ~ S1)):(interval:<0,1>:10);
(markers ~ MAP:(CONS:beta  ~ S1)):(interval:<0,1>:10);
(markers ~ MAP:(CONS:gamma ~ S1)):(interval:<0,1>:10);
```

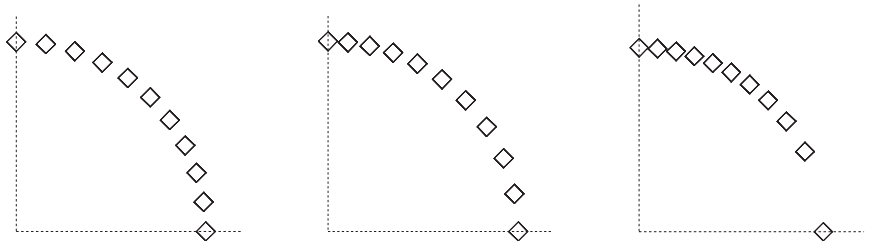

Figure 5.1 Trigonometric, rational and Cartesian parametrization

Example 5.1.5 (Coordinate maps)
It should be remembered that curves, as $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ in the previous example, are vector-valued functions. In order to obtain their coordinate maps, say $\alpha_i : \mathbb{R} \to \mathbb{R}$, a composition with the appropriate selector function is needed:

$$\alpha_i = \sigma(i) \circ \boldsymbol{\alpha}$$

The conversion from a 3D vector-valued function `curve := CONS:alpha` to the sequence of its coordinate functions may be obtained in `PLaSM` as:

```
< S1 ∼ curve, S2 ∼ curve, S3 ∼ curve >;
```

Such an approach is quite expensive and inefficient, because the curve function is repeatedly evaluated on data points to get its three component functions. So, for the sake of efficiency, we suggest maintaining a coordinate representation as a *sequence* of scalar-valued functions, and then `CONS` it into a single *vector-valued* function only when it is strictly necessary.

Anyway, an operator `Curve2MapVect` is given in Script 5.1.4. This operator, when applied to a vector-valued map $\mathbb{R} \to \mathbb{E}^d$ with arbitrary d, will return the sequence of its d coordinate functions.

Script 5.1.4 (Curve to vector of maps)

```
DEF Curve2MapVect (curve::IsFun) = AA:COMP:(selectors DISTR curve)
WHERE
   selectors = AA:SEL:(1..d),
   d = LEN:(curve:<0>)
END;

Curve2MapVect:curve ≡ < curve1 , curve2 >
```

5.1.2 Reparametrization

A *smooth curve* is defined as a curve whose coordinate functions are smooth (see Section 5.2) i.e. can be derived as many times it is needed. If $\boldsymbol{c} : I \to \mathbb{E}^d$, with $I \subset \mathbb{R}$, is a smooth curve and $\rho : I \to I$ is a smooth invertible function, i.e. a *diffeomorphism*, then also

$$\boldsymbol{c}_\rho = \boldsymbol{c} \circ \rho$$

is a smooth curve. It is called a *reparametrization* of $\boldsymbol{c}$.

A very simple reparametrization is the *change of origin*. For example $\boldsymbol{c}_\rho$ is called a change of origin when

$$\rho = \mathrm{id} + \underline{c}.$$

A reparametrization $\boldsymbol{c}_\tau$ by an affine function

$$\tau = \underline{a}\,\mathrm{id} + \underline{c},$$

with $a \neq 0$, is called an *affine reparametrization*.

Example 5.1.6 (Circle reparametrization)
The circle with the center in the origin and unit radius may be parametrized on different intervals:

$$\begin{aligned} \boldsymbol{c}_1(u) &= \begin{pmatrix} \cos u & \sin u \end{pmatrix}, & u \in [0, 2\pi] \\ \boldsymbol{c}_2(u) &= \begin{pmatrix} \cos(2\pi u) & \sin(2\pi u) \end{pmatrix}, & u \in [0, 1] \end{aligned}$$

or with a different starting point:

$$\boldsymbol{c}_3(u) \quad = \quad \left(\begin{array}{cc} \cos(2\pi u + \frac{\pi}{2}) & \sin(2\pi u + \frac{\pi}{2}) \end{array} \right), \qquad u \in [0,1]$$

The reparametrization becomes evident if we use a variable-free representation:

$$\begin{aligned} \boldsymbol{c}_1 &= \left(\begin{array}{cc} \cos & \sin \end{array} \right), \\ \boldsymbol{c}_2 &= \left(\begin{array}{cc} \cos & \sin \end{array} \right) \circ (\underline{2}\,\underline{\pi}\,\mathrm{id}), \\ \boldsymbol{c}_3 &= \left(\begin{array}{cc} \cos & \sin \end{array} \right) \circ (\underline{2}\,\underline{\pi}\,\mathrm{id} + \underline{\pi}/\underline{2}). \end{aligned}$$

Example 5.1.7 (Reparametrization of a parabola)
Several curves may have the same image. Two different parametrizations of the same subset of parabola $y = x^2$ are given below:

$$\begin{aligned} \boldsymbol{p}_1(u) &= \left(\begin{array}{cc} u & u^2 \end{array} \right), \qquad u \in [0,2], \\ \boldsymbol{p}_2(u) &= \left(\begin{array}{cc} 2u & 4u^2 \end{array} \right), \qquad u \in [0,1], \end{aligned}$$

Using the variable-free notation, we have:

$$\begin{aligned} \boldsymbol{p}_1 &= \left(\begin{array}{cc} \mathrm{id} & \mathrm{id}^2 \end{array} \right), \\ \boldsymbol{p}_2 &= \left(\begin{array}{cc} \mathrm{id} & \mathrm{id}^2 \end{array} \right) \circ \underline{2}\,\mathrm{id}. \end{aligned}$$

A very similar PLaSM implementation of curve $\boldsymbol{p}_2$ follows. The `interval` function is defined in Script 5.1.1.

Script 5.1.5 (Parabola)

```
DEF p2 = [ID, ID * ID] ~ (K:2 * S1);

MAP:p2:(interval:<0,1>:30);
```

5.1.3 Orientation

Two curves with the same image can be distinguished by the sense in which the image is traversed for increasing values of the parameter. Two curves which are traversed in the same way are said to have the same *orientation*. Actually, an orientation is an equivalence class of parametrizations.

A *reversed orientation* of a curve $\boldsymbol{c} : \mathbb{R} \to \mathbb{E}^d$, with image $\boldsymbol{c}[a,b]$, is given by the affine reparametrization $\boldsymbol{c}_\lambda = \boldsymbol{c} \circ \lambda$, where $\lambda : \mathbb{R} \to \mathbb{R}$ such that $x \mapsto -x + (a+b)$. This map can be written as:

$$\lambda = \underline{-1}\,\mathrm{id} + (\underline{a+b}).$$

Example 5.1.8 (Reversing orientation)
It is useful to have a PLaSM function REV, which reverses the orientation of any curve parametrized on the interval $[a, b] \subset \mathbb{R}$. The REV function will therefore be abstracted with respect to the bounds a and b, which are real numbers.

In Script 5.1.6 we also give a polyhedral approximation of the boundary of unit circle centered in the origin, as seen from the angle interval $[0, \frac{\pi}{2}]$. The curve is a map from $[0, 1]$ with reversed orientation.

Script 5.1.6

```
DEF REV (a,b::IsReal) = K:-1 * ID + K:(a+b);
DEF alpha = [ COS, SIN ] ~ (K:(PI/2) * ID);

MAP:(alpha ~ REV:<0,1> ~ S1):(interval:<0,1>:20);
```

5.2 Differentiation

Let $\boldsymbol{\phi} : U \to F$ be a *continuous* map, with E, F normed linear spaces, $U \subset E$ open set, and $\boldsymbol{x} \in U$.

Definition 5.2.1 *The map $\boldsymbol{\phi}$ is* differentiable *at $\boldsymbol{x}$ if there is a* linear *map $\boldsymbol{L}_x : E \to F$ which approximates $\boldsymbol{\phi}$ at $\boldsymbol{x}$, i.e. such that*

$$(\boldsymbol{\phi}(\boldsymbol{x} + \boldsymbol{h}) - \boldsymbol{\phi}(\boldsymbol{x})) - \boldsymbol{L}_{\boldsymbol{x}}(\boldsymbol{h}) = o(\|\boldsymbol{h}\|)$$

where $o(\|\boldsymbol{h}\|)$ means that the left-hand side approaches zero faster than $\|\boldsymbol{h}\|$. In general, $\boldsymbol{L}_{\boldsymbol{x}}$ depends on $\boldsymbol{x}$. When it exists, it is unique. If $\boldsymbol{\phi}$ is differentiable at every $\boldsymbol{x} \in U$ we say that $\boldsymbol{\phi}$ is differentiable in U.

$\boldsymbol{L}_x$ is a linear map in $\mathrm{lin}\,(E; F)$. It is called the *(Fréchet-)derivative of $\boldsymbol{\phi}$ at $\boldsymbol{x}$* and is denoted as $D\boldsymbol{\phi}(\boldsymbol{x})$. Hence we can write:

$$\boldsymbol{\phi}(\boldsymbol{x} + \boldsymbol{h}) - \boldsymbol{\phi}(\boldsymbol{x}) = D\boldsymbol{\phi}(\boldsymbol{x})(\boldsymbol{h}) + o(\boldsymbol{h})$$

The map $D\boldsymbol{\phi} : U \to \mathrm{lin}\,(E; F)$ is called the *derivative of $\boldsymbol{\phi}$*.

A map $\boldsymbol{\phi} : U \to F$ is *continuously differentiable* if it is differentiable and $D\boldsymbol{\phi} \in C^0(U; \mathrm{lin}\,(E, F))$. We also say that $\boldsymbol{\phi}$ is of class C^1. Analogously,

$$C^n = \{\boldsymbol{\phi} \in C^{n-1} : D\boldsymbol{\phi} \in C^{n-1}\}.$$

is the class of functions which are n-times differentiable. A *smooth* function is a function of class C^∞.

5.2.1 Real-valued maps of a real variable

For real maps of a single real variable we have $E = F = \mathbb{R}$. In this case the elementary notion of derivative can be recovered from the above definition.

The function $\phi : \mathbb{R} \to \mathbb{R}$ is said *differentiable* at x if the limit

$$\lim_{h \to 0} \frac{1}{h} (\phi(x + h) - \phi(x))$$

exists and is finite. That limit is called the *derivative* of ϕ at x, and is denoted by $\phi'(x)$. Its value λ is both a number in $\mathbb{R}$ and a linear map in $\text{lin}(\mathbb{R}, \mathbb{R})$ via the canonical isomorphism $\lambda \mapsto \lambda(1) = \lambda$, such that $y \mapsto \lambda y$.

Hence we may write for the ordinary derivative $\phi'(x) = D\phi(x)(1)$. This value is the *slope* of the tangent to the graph of ϕ at point $(x, \phi(x))$.

Because of the canonical isomorphism and the consequent identification of $\mathbb{R}$ with the dual space $\mathbb{R}^*$ of linear maps $\mathbb{R} \to \mathbb{R}$, we may also write, for $\phi : \mathbb{R} \to \mathbb{R}$

$$D\phi : \mathbb{R} \to \mathbb{R}; x \mapsto \phi'(x).$$

Implementation The value of $\phi'(x)$ can be computed, according to its definition, by the function `deriv0` of Script 5.2.1, with `h` a suitably "small" number. A *much* better numerical approximation to the slope of tangent to graph of ϕ at x is given by the central difference implemented by function `deriv1`, as shown by the following approximations of $\sin' \frac{\pi}{3} = \cos \frac{\pi}{3} = \frac{1}{2}$:

```
deriv0:1E-4:SIN:(PI/3) ≡ 0.499956697895
deriv1:1E-4:SIN:(PI/3) ≡ 0.499999999166
```

Script 5.2.1 (Derivation of $\phi : \mathbb{R} \to \mathbb{R}$ at x)

```
DEF deriv0(h::IsReal)(f::IsFun)(x::IsReal) = (f:(x + h) - f:x)/h;
DEF deriv1(h::IsReal)(f::IsFun)(x::IsReal) = (f:(x + h) - f:(x - h))/(2*h);

DEF D11 (f::IsFun)(x::IsReal) = deriv1:1E-4:f:x;
```

The `D11` operator is given to compute the derivative of functions $\mathbb{R} \to \mathbb{R}$ at a point x. Such specialized operator will be needed in Script 5.2.15 to implement a *generalized* derivation operator `D`, to be used to derive maps between spaces of any dimensions. Hence in the rest of the chapter we only make use of this generalized operator.

The differentiation operator `D` must be applied first to a function, and then to a real number, in order to return a real number. When it is applied only to a function, it returns the *derivative function* of its argument:

```
D:COS:0 ≡ 0.0
D:COS:(PI/2) ≡ -0.9999999983332231
D:COS ≡ An-Anonymous-Function
```

Example 5.2.1 (Map of the derivate functions)
In Figure 5.2 it is shown the graph of the cos function, together with the graph of the $D\cos$ function. For this purpose we apply, in Script 5.2.2, the `MAP` primitive to function `[ID, COS]` $\circ\, \sigma(1)$ and to function `[ID, D:COS]` $\circ\, \sigma(1)$, respectively, and

then to a 1D polyhedral complex which defines a partition of the domain $[-\pi, \pi]$ with 60 subintervals.

Notice that the same approach is used to generate the graphs of x and y axes, by mapping the `[ID, K:0]` $\circ\ \sigma(1)$ and `[K:0, ID]` $\circ\ \sigma(1)$ functions, respectively. The generator `interval` of the domain partition is given in Script 5.1.1.

Script 5.2.2

```
STRUCT:<
   MAP:([ID, COS] ~ S1):(interval:<-:PI,PI>:60),
   MAP:([ID, D:COS] ~ S1):(interval:<-:PI,PI>:60),
   MAP:([ID, K:0] ~ S1):(interval:<-:PI,PI>:1),
   MAP:([K:0, ID] ~ S1):(interval:<-1,1>:1) >;
```

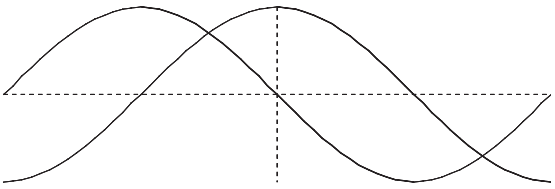

Figure 5.2 Graphs of cos and $D\cos$ functions

Higher-order derivatives

The differentiation operator D can be applied repeatedly on a smooth function $\phi : \mathbb{R} \to \mathbb{R}$. In particular:

$$D^2\,\phi := (D \circ D)\,\phi = D\,\phi' = (\phi')' =: \phi^{(2)}.$$

More in general, for each integer $n > 1$,

$$\phi^{(n)} := D^n\,\phi = D\,\phi^{(n-1)}.$$

Example 5.2.2 (Polynomials)
The set of polynomial functions of degree $\leq n$ with coefficients in $\mathbb{R}$ is often denoted as $P_n[\mathbb{R}]$. Functions $\mu \in P_n[\mathbb{R}]$, where

$$\mu : \mathbb{R} \to \mathbb{R};\ x \mapsto \sum_{i=0}^{n} a_i x^i, \qquad a_i \in \mathbb{R},$$

are an important example of *smooth functions*, and constitute a vector space of dimension $n + 1$. In later chapters we will see that some special bases of such vector space play a very important role in defining *free-form* curves and surfaces used by CAD systems.

It is very easy to see that the derivative of a polynomial of degree n is a polynomial of degree $n - 1$.

In other words, if $\mu \in P_n[\mathbb{R}]$, then $D\mu \in P_{n-1}[\mathbb{R}]$. This fact implies that

$$P_n[\mathbb{R}] \supset D(P_n[\mathbb{R}]) \supset \cdots \supset D^{n-1}(P_n[\mathbb{R}]) \supset D^n(P_n[\mathbb{R}]).$$

Example 5.2.3 (Graph of derivatives of cos**)**
The graphs of functions cos, $D\cos$, $D^2\cos$ and $D^3\cos$ on a partition of the domain $[-\pi, \pi]$ with 60 subintervals are produced by the last expression in Script 5.2.3, and are shown in Figure 5.3.

Script 5.2.3

```
DEF dom = interval:<-:PI,PI>:60;

STRUCT:<
   MAP:([ID, COS] ~ S1):dom,
   MAP:([ID, D:COS] ~ S1):dom,
   MAP:([ID, (D ~ D):COS] ~ S1):dom,
   MAP:([ID, (D ~ D ~ D):COS] ~ S1):dom >;
```

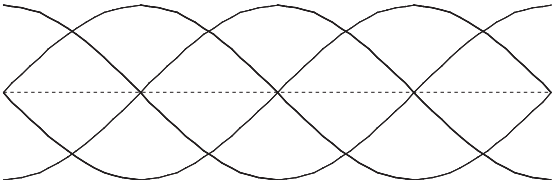

Figure 5.3 Graphs of cos function and of its first three derivatives

Useful properties of derivatives

Let us consider $E = F = \mathbb{R}$. The following properties hold for maps in $C^1(\mathbb{R};\mathbb{R})$, in the following denoted as C^1. Actually, such properties hold also in more general settings.

Linearity For all $a, b \in \mathbb{R}$ and all $\phi, \psi \in C^1$, the *linear combination* map $a\phi + b\psi \in C^1$, with

$$D(a\phi + b\psi) = aD\phi + bD\psi.$$

Chain rule For all $\phi, \psi \in C^1$, the *compound* map $\phi \circ \psi \in C^1$, with

$$D(\phi \circ \psi) = (D\phi \circ \psi)D\psi.$$

Leibnitz's rule For all $\phi, \psi \in C^1$, the *product* map $\phi\psi \in C^1$, with

$$D(\phi\psi) = (D\phi)\psi + \phi(D\psi).$$

Example 5.2.4
In Figure 5.4 we show the graph of $\sin \circ \cos$ function and of its three first derivatives $D(\sin \circ \cos)$, $D^2(\sin \circ \cos)$ and $D^3(\sin \circ \cos)$, as generated by Script 5.2.4.

Script 5.2.4 (A few derivatives of $\sin \circ \cos$**)**

```
STRUCT:<
   MAP:([ID, (SIN ~ COS)] ~ S1):dom,
   MAP:([ID, D:(SIN ~ COS)] ~ S1):dom,
   MAP:([ID, (D ~ D):(SIN ~ COS)] ~ S1):dom,
   MAP:([ID, (D ~ D ~ D):(SIN ~ COS)] ~ S1):dom
>;
```

The reader may check that the same graph generated by Script 5.2.4 and shown in Figure 5.4 can be produced by substituting the right-hand side of the following expressions to the left-hand side. The first equivalence is produced by the chain rule; the second equivalence is generated by using both chain and Leibnitz's rules.

```
D:(SIN ~ COS) ≡ (D:SIN ~ COS) * D:COS

(D ~ D):(SIN ~ COS) ≡
    D:(D:SIN ~ COS) * D:COS + (D:SIN ~ COS) * D:(D:COS)
```

An equivalent symbolic expression for `(D ~ D ~ D):(SIN ~ COS)` would produce several lines of code, as the diligent reader might check. Furthermore, it would require a longer computation and would produce a greater numeric error when evaluated on actual data.

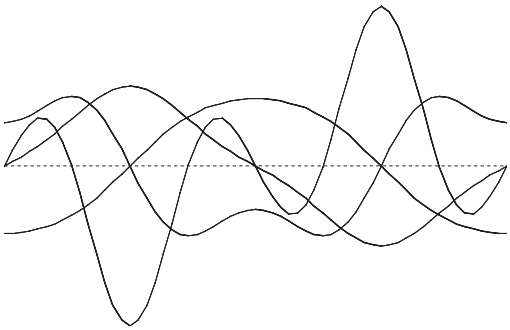

Figure 5.4 Graphs of $\phi = \sin \circ \cos$ and of its first three derivatives on $[-\pi, \pi]$

5.2.2 Vector-valued maps of a real variable

In this case we have $E = \mathbb{R}$, $F = \mathbb{R}^m$, and $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_m)$ is a vector-valued *continuous* map $\mathbb{R} \to \mathbb{R}^m$. In other words, $\boldsymbol{\phi} \in C^0(\mathbb{R}; \mathbb{R}^m)$. Such a map is called a *curve* in $\mathbb{R}^m$.

At each $u \in \mathbb{R}$ the derivative $D\boldsymbol{\phi}(u)$, if it exists, is a *linear* map $\mathbb{R} \to \mathbb{R}^m$.

By linearity it is, for all $s \in \mathbb{R}$:

$$D\boldsymbol{\phi}(u)(s) = D\boldsymbol{\phi}(u)(s1) = sD\boldsymbol{\phi}(u)(1) = s\boldsymbol{\phi}'(u) = s(\phi_1'(u), \ldots, \phi_m'(u))$$

Implementation The function `D1m`, given in Script 5.2.5, which implements this derivative, is very easy to write down by considering that $D\boldsymbol{\phi}(u)(s)$ is just a scalar multiple of the vector $(\phi_1'(u), \ldots, \phi_m'(u))$ whose components are the derivatives at u of the scalar functions $\phi_1, \ldots, \phi_m \in C^1(\mathbb{R}; \mathbb{R})$.

Notice that the `scalarVectProd` operator, defined in Script 2.1.20 and normally used to multiply a scalar number times an ordered d-tuple of scalars, is here used to multiply a scalar function times an ordered d-tuple of functions. Such useful behavior of vector operators will often be found in this chapter.

Script 5.2.5 (Derivative of $f : \mathbb{R} \to \mathbb{R}^m$ at u)

```
DEF D1m (f::IsSeqOf:IsFun)(u::IsReal) =
   S1 scalarVectProd AA:K:((CONS ~ AA:D11): f: u);
```

In the following examples, according to Script 5.2.15, we use the generalized operator `D` instead of the specialized operator `D1m`.

Notice also that in the remainder of this chapter we usually implement in `PLaSM` a curve $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_m)$ as the *sequence*

`<` ϕ_1`, ... ,` ϕ_m `>`

of its coordinate functions, and not as the vector-valued function

`[` ϕ_1`, ... ,` ϕ_m `]`

given by their `CONS`. Such implementation choice allows for easier access and manipulation of the coordinate functions, and is a consequence of the discussion in Example 5.1.5.

Example 5.2.5 (Circular curve)
The circular curve of unit radius centered at the origin may be given as a map

$$\texttt{circlemap} : \mathbb{R} \to \mathbb{R}^2;\ u \mapsto (\cos u, \sin u), \qquad u \in [-\pi, \pi]$$

The derivative $D\texttt{circlemap}(u)$ at u is a linear map. In Figure 5.5a we show both the image curve $\texttt{circlemap}[-\pi, \pi]$ and the line segment $D\texttt{circlemap}(\frac{\pi}{3})[-\frac{1}{2}, \frac{1}{2}]$. Notice that the point $D\texttt{circlemap}(\frac{\pi}{3})(0)$ clearly coincides with $(0, 0)$.

The last expression of Script 5.2.6 produces Figure 5.5b, where the set $D\texttt{circlemap}(\frac{\pi}{3})[-\frac{1}{2}, \frac{1}{2}]$ has been translated to the tangency point $\texttt{circlemap}(\frac{\pi}{3})$. The part of tangent line there shown is exactly the point set

$$\texttt{circlemap}(\frac{\pi}{3}) + D\texttt{circlemap}(\frac{\pi}{3})[-\frac{1}{2}, \frac{1}{2}].$$

where, as usual, $f(A)$ stands for $\{f(a) \mid a \in A\}$, and $x + B$ stands for $\{x + b \mid b \in B\}$.

Script 5.2.6 (Derivative at a circle point)

```
DEF circlemap = < COS, SIN >;

STRUCT:<
   MAP:(CONS:circlemap ~ s1):(interval:<-:PI,PI>:60),
   T:<1,2>:(CONS:circlemap:(PI/3)),
   MAP:(CONS:(D:circlemap:(PI/3))):(interval:<-0.5,0.5>:1) >;
```

Figure 5.5 (a) Derivative of `circlemap` map $\mathbb{R} \to \mathbb{R}^2$ at $\frac{\pi}{3}$ (b) Corresponding affine map

Example 5.2.6 (Circular helix)
The *circular helix* curve of radius r, pitch h and n number of 2π turns can be given as the map

$$\texttt{helix}(r,h,n) : \mathbb{R} \to \mathbb{R}^3 : u \mapsto (r\cos u, r\sin u, \frac{h}{2\pi}u), \qquad u \in [0, 2\pi n].$$

The corresponding variable-free formulation is

$$\texttt{helix}(r,h,n) = \left(\begin{array}{cccc} \underline{r}\cos & \underline{r}\sin & \underline{(\frac{h}{2\pi})}\mathrm{id} \end{array} \right)$$

The collection of helix curves with axis on $x = y = 0$, starting point on the x-axis, and prescribed radius, height and number of turns, may hence be generated by the `PLaSM` function

$$\texttt{helix} : \mathbb{R}^+ \times \mathbb{R}^+ \times \mathbb{R}^+ \to (\mathbb{R} \to \mathbb{R}^3)$$

which is implemented in Script 5.2.7 as a sequence of three coordinate functions.

Script 5.2.7 (Derivative at a helix point)

```
DEF helix (r,h,n::IsRealPos) = < K:r*COS, K:r*SIN, K:(h/(2*PI))*ID >;

STRUCT:<
   MAP:(CONS:(helix:<1,0.5,4>) ~ s1):(interval:<0,8*PI>:180),
   T:<1,2,3>:(CONS:(helix:<1,0.5,4>):(2*PI/3)),
   MAP:(CONS:(D:(helix:<1,0.5,4>):(2*PI/3))):(interval:<-0.5,0.5>:1)
>;
```

The `STRUCT` expression gives both the helix image of the interval $[0, 8\pi]$ and the image of the tangent map in $\frac{2}{3}\pi$ on the interval $[-\frac{1}{2}, \frac{1}{2}]$. To be precise, the `STRUCT`

expression of Script 5.2.7 produces (a simplicial approximation of) the sets

$$\texttt{helix}(r,h,n)[0,8\pi] \qquad \text{and}$$
$$\texttt{helix}(r,h,n)(\frac{2}{3}\pi) + D\texttt{helix}(r,h,n)(\frac{2}{3}\pi)[-\frac{1}{2},\frac{1}{2}]$$

with $r = 1$, $h = 0.5$ and $n = 4$. Both such sets are shown in Figure 5.6.

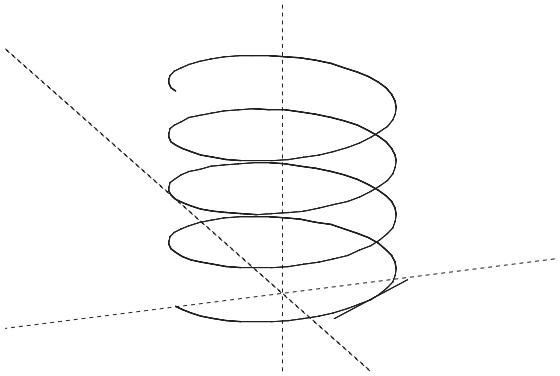

Figure 5.6 Image of the tangent map to the `helix` curve at the point $\frac{2}{3}\pi$

5.2.3 Real-valued maps of several real variables

In this case we have $E = \mathbb{R}^n$ and $F = \mathbb{R}$, where $\phi : \mathbb{R}^n \to \mathbb{R}$ is a real-valued function of n real variables. The derivative $D\phi(\boldsymbol{a})$ at $\boldsymbol{a}$ (if it exists), is a *linear functional* $\mathbb{R}^n \to \mathbb{R}$, called the *gradient* of ϕ at $\boldsymbol{a}$, and often denoted by $\nabla\phi(\boldsymbol{a})$.

In order to compute $D\phi(\boldsymbol{a})$, notice that each $\boldsymbol{h} \in \mathbb{R}^n$ can be written, using the standard basis $(\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n)$, as $\boldsymbol{h} = h_1\boldsymbol{e}_1 + \cdots + h_n\boldsymbol{e}_n$, so that, by linearity

$$D\phi(\boldsymbol{a})(\boldsymbol{h}) = h_1 Df(\boldsymbol{a})(\boldsymbol{e}_1) + \cdots + h_n Df(\boldsymbol{a})(\boldsymbol{e}_n).$$

Each $Df(\boldsymbol{a})(\boldsymbol{e}_i)$ is the limit of the ratio of the function difference in two close points $\boldsymbol{a} - dx_i\boldsymbol{e}_i$ and $\boldsymbol{a} + dx_i\boldsymbol{e}_i$, over their distance $2\,dx_i$, but this limit of the "incremental ratio" is exactly the derivative of f at $\boldsymbol{a}$ as a function of a single real variable, usually known as the *partial derivative* $\frac{\partial\phi}{\partial x_i}(\boldsymbol{a}) \in \mathrm{lin}\,(\mathbb{R}^n; \mathbb{R})$, so that we can write:

$$D\phi(\boldsymbol{a})(\boldsymbol{h}) = h_1\frac{\partial\phi}{\partial x_1}(\boldsymbol{a}) + \cdots + h_n\frac{\partial\phi}{\partial x_n}(\boldsymbol{a}) = \nabla\phi(\boldsymbol{a}) \cdot \boldsymbol{h}.$$

Implementation To compute a partial derivative `Dp` of ϕ at $\boldsymbol{a}$ in the i-th coordinate direction, it is necessary to compute the function difference in two points `a1` and `a2` which differ only for a suitably "small" amount $2\,dx$ of the i-th coordinate, if we choose a numerical scheme based on central differences.

The *gradient* at $\boldsymbol{a}$ of a function $\phi : \mathbb{R}^n \to \mathbb{R}$ is the column vector of the partial derivatives of ϕ at $\boldsymbol{a}$. It is easily implemented in Script 5.2.9.

Some examples of use of the `grad` operator are given in Script 5.2.9. They may help to understand the semantics of the `grad` implementation.

Script 5.2.8 (Partial derivative of $f : \mathbb{R}^n \to \mathbb{R}$ at a)

```
DEF Dp (i::isIntPos)(f::IsFun)(a::IsPoint)(h::IsVect) =
   (f:a2 - f:a1) / (2*dx)
WHERE
   a2 = CAT:<seq1, <SEL:i:a + dx>, seq2> ,
   a1 = CAT:<seq1, <SEL:i:a - dx>, seq2> ,
   seq1 = AS:SEL:(1..(i - 1)):a ,
   seq2 = AS:SEL:((i + 1)..n):a ,
   n = LEN:a ,
   dx = 1E-5
END;
```

Script 5.2.9 (Gradient of $\phi : \mathbb{R}^n \to \mathbb{R}$ at a)

```
DEF grad (f::IsFun)(a::IsPoint) = CONS:(DpVect:f):a
WHERE
   DpVect = (CONS ~ AA:Dp ~ INTSTO ~ LEN):a
END;

DEF f = SIN ~ S1 * SIN ~ S2;

grad:f ≡ An-Anonymous-Function
grad:f:<PI/3,PI/-2> ≡ < An-Anonymous-Function, An-Anonymous-Function >
CONS:(grad:f:<PI/3,PI/-2>):<1,1> ≡ <-0.4999999999921733, 0.0>
```

It may be also useful to show the variable-free representation of the map generated when using the `Dn1` operator given in Script 5.2.10. Notice first of all, that such a function, like most `PLaSM` programs, is dimension-independent and works for any dimension n of the domain space.

So, we have

$$\texttt{Dn1}(\phi)(\boldsymbol{a}) := \frac{\partial \phi}{\partial x_1}(\boldsymbol{a})\sigma(1) + \cdots + \frac{\partial \phi}{\partial x_n}(\boldsymbol{a})\sigma(n),$$

where $\sigma(1), \ldots, \sigma(n)$ are the selector functions defined in Section 5.1.1, and

$$\texttt{Dn1}(\phi)(\boldsymbol{a})(\boldsymbol{u}) = \frac{\partial \phi}{\partial x_1}(\boldsymbol{a})u_1 + \cdots + \frac{\partial \phi}{\partial x_n}(\boldsymbol{a})u_n = \nabla\phi(\boldsymbol{a}) \cdot \boldsymbol{u}.$$

Script 5.2.10 (Derivative of $\phi : \mathbb{R}^n \to \mathbb{R}$ at $\boldsymbol{a}$)

```
DEF Dn1 (f::IsFun)(a::IsPoint) =
   (InnerProd ~ [AA:SEL ~ INTSTO ~ LEN, grad:f]):a ;
```

The reader should notice that the `PLaSM` expression

```
(AA:SEL ~ INTSTO ~ LEN):a
```

returns the sequence `< σ(1), ... , σ(n) >` of selector functions, as well as the expression `grad:f:a` returns the sequence of partial derivatives of `f` at `a`. Also notice

that the binary `InnerProd` operator, defined in Script 3.2.4, works with pairs of ordered d-tuples of either scalar numbers or scalar functions.

As always, we will use in the remainder of this chapter the generalized operator `D` of Script 5.2.15, instead of the specialized derivation operator `Dn1`.

Directional derivative The weaker concept of *directional* (or *Gateaux*) *derivative* is defined as follows. Let $\boldsymbol{h} \in \mathbb{R}^n$ be a unit vector and

$$\psi : \mathbb{R} \to \mathbb{R};\ t \mapsto \phi(\boldsymbol{a} + t\boldsymbol{h}).$$

Then the directional derivative of ϕ in the chosen direction $\boldsymbol{h}$ is the scalar $\psi'(0) = D\psi(0)(1)$.

This directional derivative may be interpreted as the derivative along any curve passing for $\boldsymbol{a}$ with $\boldsymbol{h}$ as tangent vector at $\boldsymbol{a}$:

$$D_{\boldsymbol{h}}\phi(\boldsymbol{a}) = \lim_{t \to 0} \frac{\phi(\boldsymbol{a} + t\boldsymbol{h})}{t}$$

The map ϕ is said to be *Gateaux-differentiable* at $\boldsymbol{a}$ if $D_{\boldsymbol{h}}\phi(\boldsymbol{a})$ exists for all $\boldsymbol{h} \in \mathbb{R}^n$. When $D\phi(\boldsymbol{a})$ exists, $D_{\boldsymbol{h}}\phi(\boldsymbol{a}) = D\phi(\boldsymbol{a})(\boldsymbol{h}) = \nabla\phi(\boldsymbol{a}) \cdot \boldsymbol{h}$.

Example 5.2.7 (Graph of tangent space)
The graphs of $\sin u \sin v$, i.e. of the function $\phi = (\sin \circ \sigma(1))(\sin \circ \sigma(2))$ on the set $U_1 = [0, \pi]^2$, and of the affine map $(\phi + D\phi)(\frac{\pi}{3}, \frac{\pi}{2})$ on the set $U_2 = [-\frac{1}{2}, \frac{1}{2}]^2$, is produced by Script 5.2.11 and is shown in Figure 5.7.

Script 5.2.11 (Graph of tangent space)

```
DEF f  = SIN ~ S1 * SIN ~ S2;
DEF U1 = interval2D:<0,0,PI,PI>:<20,20>;
DEF U2 = interval2D:<-0.5,-0.5,0.5,0.5>:<2,2>;

STRUCT:<
   MAP:[S1, S2, f]:U1 CREASE (PI/2),
   T:<1,2,3>:([S1, S2, f]:<PI/3, PI/2>),
   MAP:[S1, S2, D:f:<PI/3, PI/2>]:U2 COLOR RED
>;
```

Within the above PLaSM code two rendering-oriented language constructs are informally introduced. The `CREASE` binary operator, applied to a polyhedral complex and to an angle (in radiants), annotates the first argument with a rendering property checked by the VRML exporter. The resulting effect is the exporting of the surface with "indexed coordinates", so that a "Gouraud's shader" can be applied by the VRML browser in rendering the surface. Analogously, the binary operator `COLOR` applies the predefined RGB constant `RED` to its polyhedral argument. Both the `CREASE` and the `COLOR` operators require the loading of the PLaSM library named "`colors`". Such topics are discussed in detail in Chapter 10.

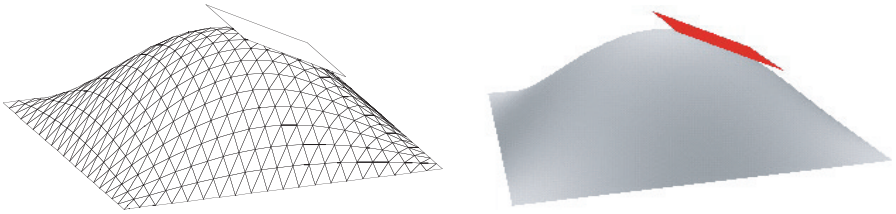

Figure 5.7 Graphs of map ϕ on $[0,\pi]^2$ and of map $(\phi + D\phi)(\frac{\pi}{3},\frac{\pi}{2})$ on $[-1,1]^2$, with $\phi = \sin u \sin v$: (a) polyhedral approximation (b) smooth rendering

In more precise terms, the `STRUCT` expresssion of Script 5.2.11 generates a simplicial approximation of the set:

$$(\sigma(1),\sigma(2),\phi)[0,\pi]^2 \cup \Big((\sigma(1),\sigma(2),D\phi(\frac{\pi}{3},\frac{\pi}{2}))[-1,1]^2 + \phi(\frac{\pi}{3},\frac{\pi}{2})\Big)$$

5.2.4 Vector-valued maps of several real variables

In this case we have $E = \mathbb{R}^n, F = \mathbb{R}^m$ and $\boldsymbol{\phi} : \mathbb{R}^n \to \mathbb{R}^m$. The derivative $D\boldsymbol{\phi}(\boldsymbol{u})$ of $\boldsymbol{\phi}$ at $\boldsymbol{u}$ is a *linear map* $\mathbb{R}^n \to \mathbb{R}^m$, if it exists.

In other words, $D\boldsymbol{\phi}(\boldsymbol{u})$ is a matrix in $\mathbb{R}^m_n$, called *Jacobian matrix* of $\boldsymbol{\phi}$ at $\boldsymbol{u}$.

By chosing bases (e_i) in $\mathbb{R}^n$ and (ϵ_j) in $\mathbb{R}^m$, with $\boldsymbol{\phi} = (\phi_1,\phi_2,\ldots,\phi_m)$ and each $\phi_i : \mathbb{R}^n \to \mathbb{R}$, we have

$$D\boldsymbol{\phi}(\boldsymbol{u}) = D\phi_1(\boldsymbol{u})\epsilon_1 + D\phi_2(\boldsymbol{u})\epsilon_2 + \cdots + D\phi_m(\boldsymbol{u})\epsilon_m,$$

The Jacobian matrix $D\boldsymbol{\phi}(\boldsymbol{u})$ is often represented as

$$\frac{\partial(\phi_1,\phi_2,\ldots,\phi_m)}{\partial(x_1,x_2,\ldots,x_n)} = (\alpha_{ij}), \quad \text{with} \quad \alpha_{ij} = \frac{\partial\phi_i}{\partial x_j}(x).$$

The determinant

$$J = \left|\frac{\partial(\phi_1,\phi_2,\ldots,\phi_m)}{\partial(x_1,x_2,\ldots,x_n)}\right|$$

of the Jacobian matrix at $\boldsymbol{u}$ is called the *Jacobian* of $\boldsymbol{\phi}$ at $\boldsymbol{u}$. When $\boldsymbol{\phi} \in C^1(\mathbb{R}^n;\mathbb{R}^n)$ is a transformation of coordinates, $J(\boldsymbol{u})$ gives the ratio of the volume elements at $\boldsymbol{u}$, with $dV' = J(\boldsymbol{u})dV$.

Implementation Once again, this derivative is quite easy to implement in `PLaSM`. According to the functional character of the language, the easiest implementation of $D\boldsymbol{\phi}(\boldsymbol{u})$ is as the sequence $(D\phi_1(\boldsymbol{u}),\ldots,D\phi_m(\boldsymbol{u}))$ of derivatives at $\boldsymbol{u}$ of the coordinate functions of $\boldsymbol{\phi}$. Such a derivative is denoted as `Dnm` in Script 5.2.12.

Script 5.2.12 (Derivative of $f : \mathbb{R}^n \to \mathbb{R}^m$ at a)

```
DEF Dnm (f::IsSeqOf:IsFun)(u::IsPoint) = (CONS ~ AA:Dn1):f:u;
```

As is usual at this point, we will refer to this operator as `D`, according to the generalized implementation of the derivative operator given in Script 5.2.15.

Example 5.2.8 (Tangent plane at a point of a torus)
The parametric equations of the family of surfaces known as *toruses* can be seen as generated by a map

$$\boldsymbol{\phi} : \mathbb{R}^+ \times \mathbb{R}^+ \to (\mathbb{R}^2 \to \mathbb{R}^3) : (r, R) \mapsto (\phi_x, \phi_y, \phi_z),$$

where r, R are the *minor* and *major* radiuses of the surfaces, respectively, and

$$\phi_x, \phi_y, \phi_z : U \to \mathbb{R},$$

with

$$\begin{aligned} \phi_x &= (\underline{r} \cos \circ\, \sigma(1) + \underline{R}) \cos \circ\, \sigma(2), \\ \phi_y &= (\underline{r} \cos \circ\, \sigma(1) + \underline{R}) \sin \circ\, \sigma(2), \\ \phi_z &= \underline{r} \sin \circ\, \sigma(1), \end{aligned}$$

and

$$U = [0, 2\pi)^2 \subset \mathbb{R}^2.$$

Such equations can be easily derived from the general equation of rotational surfaces given in Chapter 12. They are directly translated into the PLaSM definition given in Script 5.2.13.

Script 5.2.13 (Torus)

```
DEF torus (r1,r2::IsReal) = < fx, fy, fz >
WHERE
   fx = (K:r1 * COS ~ S1 + K:r2) * COS ~ S2,
   fy = (K:r1 * COS ~ S1 + K:r2) * SIN ~ S2,
   fz = (K:r1 * SIN ~ S1)
END;
```

Some preliminary check on the types of elementary expressions involving the `torus` function may be useful for a deeper understanding. In particular, we check that

1. the expression `torus:<1,3>` returns a triplet of (coordinate) functions;
2. such functions, evaluated at the same 2D point return a 3D point;
3. the derivative of `torus:<1,3>` at a point returns a triplet of (linear) functions,

as shown by the following examples:

```
torus:<1,3> ≡
   <An-Anonymous-Function, An-Anonymous-Function, An-Anonymous-Function>

CONS:(torus:<1,3>):<PI/3,PI/3> ≡
   <1.75, 3.031088913245535, 0.8660254037844386>

D:(torus:<1,3>):<PI/3,PI/3> ≡
   <An-Anonymous-Function, An-Anonymous-Function, An-Anonymous-Function>
```

Figure 5.8 Image of the affine map $(\boldsymbol{\phi} + D\boldsymbol{\phi})(\boldsymbol{u})$ on the torus surface

In Figure 5.8 we show the image of the function `torus:<1,3>` on the domain $[0, 2\pi]^2$, i.e. the torus surface with radiuses $(1, 3)$, and the image of the affine map $\mathbb{R}^2 \to \mathbb{R}^3$ of type $(\boldsymbol{\phi} + D\boldsymbol{\phi})(\boldsymbol{u})$ on the domain $[-\frac{1}{2}, \frac{1}{2}]^2$.

More formally, we can say that Figure 5.8 gives a picture of the set

$$\boldsymbol{\phi}[0, 2\pi]^2 \ \cup \ (\boldsymbol{\phi}(\boldsymbol{u}) + D\boldsymbol{\phi}(\boldsymbol{u})[-\tfrac{1}{2}, \tfrac{1}{2}]^2)$$

with $\boldsymbol{\phi} =$ `torus:<1,3>` and $\boldsymbol{u} = (\frac{\pi}{3}, \frac{\pi}{3})$.

Script 5.2.14 (Image of the affine map $(\boldsymbol{\phi}(\boldsymbol{u}) + D\boldsymbol{\phi}(\boldsymbol{u})\boldsymbol{v})$

```
STRUCT:<
   MAP:(CONS:(torus:<1,3>)):(interval2D:<0,0,2*PI,2*PI>:<24,48>) ,
   T:<1,2,3>:(CONS:(torus:<1,3>):<PI/3,PI/3>),
   MAP:(CONS:(D:(torus:<1,3>):<PI/3,PI/3>)):
      (interval2D:<-0.5,-0.5,0.5,0.5>:<2,2>) COLOR RGB:<1,0,0> >;
```

Actually, the implementation in Script 5.2.14 is based on the graphics approach of translating the image of the linear map $D\boldsymbol{\phi}(\boldsymbol{u})$ with translation vector $\boldsymbol{\phi}(\boldsymbol{u})$. A direct implementation of the affine mapping

$$\boldsymbol{v} \mapsto \boldsymbol{\phi}(\boldsymbol{u}) + D\boldsymbol{\phi}(\boldsymbol{u})\boldsymbol{v}, \qquad u = \left(\frac{\pi}{3}, \frac{\pi}{3}\right), \boldsymbol{v} \in \left[-\frac{1}{2}, \frac{1}{2}\right]^2,$$

should instead be written as

```
MAP:(vectSum
~CONS:([K~CONS:(torus:<1,3>), CONS~D:(torus:<1,3>)]:<PI/3,PI/3>)):
   (interval2D:<-0.5,-0.5,0.5,0.5>:<1,1>)
```

Notice that the greatest care was used in `CONS`ing some sub-expressions, depending on whether they return either a single vector function or a sequence of functions. In fact a sequence *cannot* be applied to any argument nor composed with functions.

5.2.5 Generalized implementation

An implementation of a derivation operator

$$\mathtt{D} : C^1(\mathbb{R}^n; \mathbb{R}^m) \to \mathrm{lin}\,(\mathbb{R}^n; \mathbb{R}^m)$$

at point $\boldsymbol{u} \in \mathbb{R}^n$ is given in this section. This operator will work for arbitrary values of n and m. To this purpose, there are four different cases that we need to manage, and for which we have already given specialized operators, respectively named `D11`, `D1m`, `Dn1` and `Dnm`.

A cascaded `IF` sequence is just used to select the proper specialized operator, to be chosen depending on the type of input parameters `f` and `u`. Notice that the `TT := K:True` predicate returns *true* for any input.

Script 5.2.15 (Generalized derivation operator)

```
DEF D (f::TT)(u::TT) =
   IF:< AND ~ [ IsFun~S1, IsReal~S2 ], APPLY ~ [D11 ~ S1,S2],
   IF:< AND ~ [ IsSeqOf:IsFun~S1, IsReal~S2 ], APPLY ~ [D1m ~ S1,S2],
   IF:< AND ~ [ IsFun~S1, IsPoint~S2 ], APPLY ~ [Dn1 ~ S1,S2],
   IF:< AND ~ [ IsSeqOf:IsFun~S1, IsPoint~S2 ], APPLY ~ [Dnm ~ S1,S2],
ID >>>>:< f, u >;
```

5.3 Fields and differential operators

A *function* $\boldsymbol{\phi} : \mathbb{E} \to F$ that maps points of an Euclidean space $\mathbb{E}$ into a linear space F is called a **field**. When F is the set of real numbers, or a vector or tensor space, $\boldsymbol{\phi}$ is called a *scalar*, *vector* or *tensor field*, respectively.

5.3.1 Gradient

The **gradient** $D\phi$ of a field $\phi : \mathbb{E} \to F$ is the map on $\mathbb{E}$ whose value at x is the *derivative* $D\phi(x)$ of ϕ at x:

$$D\phi : \mathbb{E} \to \mathrm{lin}\,(\mathbb{E}, F) : x \mapsto D\phi(x)$$

In the remainder of the chapter, we identify the Euclidean n-space $\mathbb{E}^n$ with its linear support space $\mathbb{R}^n$, and denote the gradient of a field ϕ by $\nabla\phi$.

Example 5.3.1 (Gradient of a scalar field $\mathbb{R}^m \to \mathbb{R}$)
A vector field representation of gradient $\nabla\phi$ of a scalar field as a set of line segments is produced by the three expressions in Script 5.3.1. In particular, each expression aggregates the set of vectors on the vertices of the domain discretization named `dom` and the domain itself. The scalar fields shown in Figure 5.9 are, respectively:

$$\begin{aligned}
(x, y) &\mapsto (x^2 + y^2)/a^2 && \text{with } a = \sqrt{2}, \text{ and } \texttt{dom} = [-1, 1]^2 \\
(x, y) &\mapsto (x^2 - y^2)/a^2 && \text{with } a = \sqrt{2}, \text{ and } \texttt{dom} = [-1, 1]^2 \\
(x, y) &\mapsto \sin x \ \sin y && \text{with } \texttt{dom} = [0, \pi]^2
\end{aligned}$$

The first two fields correspond to the explicit representation of quadrics surfaces called *elliptic* and *hyperbolic paraboloid*, respectively. The function `interval2D`, to generate a cell decomposition of the Cartesian product of two line intervals, was given in Script 5.1.1.

Script 5.3.1 (Vector field representation of the gradient of a scalar field)

```
(STRUCT ~ [GradMap,s2]):<(s1*s1 + s2*s2)/K:2, interval2D:<-1,-1,1,1>:<7,7>>
(STRUCT ~ [GradMap,s2]):<(s1*s1 - s2*s2)/K:2, interval2D:<-1,-1,1,1>:<7,7>>
(STRUCT ~ [GradMap,s2]):<sin ~ s1 * sin ~ s2, interval2D:<0,0,PI,PI>:<7,7>>
```

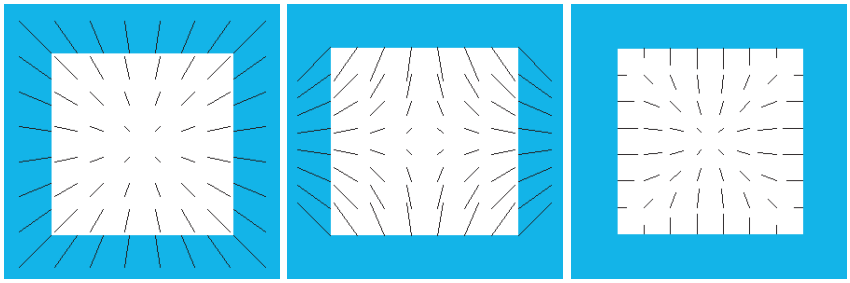

Figure 5.9 Graph of gradient of fields: (a) $\phi_1(x,y) = (x^2+y^2)/a^2$ (b) $\phi_2(x,y) = (x^2-y^2)/a^2$ (c) $\phi_3(x,y) = \sin x \ \sin y$

Implementation As we already know, if ϕ is a scalar field $\mathbb{E}^n \to \mathbb{R}$, then $D\phi$ — in the following denoted as $\nabla\phi$ — can be seen as the vector field such that for each $\boldsymbol{u} \in \mathbb{R}^n$

$$D\phi(\boldsymbol{x})(\boldsymbol{u}) = \nabla\phi(\boldsymbol{x}) \cdot \boldsymbol{u}.$$

So, in Script 5.3.2 we give a function `GradMap` to generate a map of such a vector field for every scalar function `f` defined on an arbitary (polyhedral) domain `dom`. In particular, an expression like `GradMap:< f, dom >` will generate a set of line segments attached to vertices of the cell decomposition of `dom`, where each segment at point $\boldsymbol{x}$ is a scaled image of the vector

$$\nabla f(\boldsymbol{x}) = \left(\begin{array}{ccc} \frac{\partial f}{\partial x_1}(\boldsymbol{x})(1) & \cdots & \frac{\partial f}{\partial x_n}(\boldsymbol{x})(1) \end{array} \right),$$

translated at $\boldsymbol{x}$. For this purpose a function `segment` is given in Script 5.3.2, that generates the image of vector `b - a` applied to `a`, and scaled with coefficient `sx`. The segment function makes use of the `polyline` primitive defined in Script 7.2.3.

Script 5.3.2 (Graph of gradient of a scalar field)

```
DEF Gradient (f::IsFun)(x::IsPoint) = CONS:(grad:f:x):(#:(LEN:x):1);

DEF Segment (sx::IsReal)(a,b::IsPoint) =
   (T:ind:a ~ S:ind:(#:n:sx) ~ T:ind:(AA:-:a) ~ polyline):<a,b>
WHERE
   n = LEN:a, ind = INTSTO:n
END;

DEF GradMap (f::IsFun; dom::IsPol) = (STRUCT
   ~ AA:(Segment:0.35 ~ [ID,vectSum ~ [ID,Gradient:f]])
   ~ S1 ~ UKPOL):dom
```

The simple algorithm used in implementing the `GradMap` function can be described

as follows. The `vectSum` function was given in Script 2.1.19.

1. Compute the vertices of the cells of the polyhedral decomposition of `dom`, by "unpacking" its data structure and extracting their set with the `S1` selector.
2. Apply to all such vertices the function

   ```
   segment:0.35 ~ [ID, vectSum ~ [ID, Gradient:f]]
   ```

   which generates a sequence of segments — scaled with coefficient 0.35 — with first point in each cell vertex.
3. Aggregate all such segments in the output polyhedral complex.

Gradient of a vector field The implementation of Script 5.3.2 works only for scalar fields. The gradient of a *vector field* $\boldsymbol{\phi} : \mathbb{R}^2 \to \mathbb{R}^2$ would return a tensor (represented by the 2×2 Jacobian matrix) at each point. A graphical representation of the output tensor field would be much harder to devise. Anyway, a possible implementation of a function, which returns the gradient of a vector field in a point as the Jacobian matrix in that point, is given by function `Jacobian` in Script 5.3.3.

As a check on the implementation, we compute directly $\nabla\boldsymbol{\phi}(0.25, 0.3)$, with

$$\boldsymbol{\phi}(x, y) = \left(\frac{x^2 - y^2}{2}, \frac{x^2 + y^2}{2} \right).$$

In this case we have

$$\nabla\boldsymbol{\phi}(x, y) = \begin{pmatrix} x & -y \\ x & y \end{pmatrix}, \quad \text{and} \quad \nabla\boldsymbol{\phi}(0.25, 0.3) = \begin{pmatrix} 0.25 & -0.3 \\ 0.25 & 0.3 \end{pmatrix}.$$

As we are now used to doing, the field to be differentiated must be written using the variable-free notation

$$\boldsymbol{\phi} = \left(\frac{\sigma(1)^2 - \sigma(2)^2}{\underline{2}}, \frac{\sigma(1)^2 + \sigma(2)^2}{\underline{2}} \right).$$

Script 5.3.3 (Jacobian of a vector field)

```
DEF Jacobian (f::IsSeqOf:IsFun)(x::IsPoint) = CONS:(AA:Gradient:f):x;

Jacobian:< (s1*s1 - s2*s2)/K:2, (s1*s1 + s2*s2)/K:2 >:<0.25,0.3> ≡
   <<0.24999999999995512, -0.2999999999999531>,
    <0.24999999999997247, 0.2999999999998837>> ;
```

5.3.2 Divergence and Laplacian

The divergence and Laplacian operators are defined for vector and scalar fields, respectively, and are both functions of the gradient.

The *divergence* of a vector field $\boldsymbol{\phi}$ is defined as the *trace of the gradient*:

$$\text{div } \boldsymbol{\phi} := \text{tr } \nabla\boldsymbol{\phi}$$

Notice that the gradient of a vector field is a tensor field, whereas the divergence of a vector field is a scalar field.

The *Laplacian* $\Delta\phi$ of a scalar field ϕ is defined as the *divergence of its gradient*:

$$\Delta\phi := \text{div } \nabla\phi = \text{tr } \nabla\nabla\phi = \text{tr } \nabla^2\phi$$

Accordingly with the above remark, notice that the Laplacian of a scalar field is a scalar field.

5.3.3 Curl

Let $\boldsymbol{\phi}$ be a smooth vector field. The *curl* of $\boldsymbol{\phi}$, denoted $\text{curl}\,\boldsymbol{\phi}$, is the unique vector field such that

$$(\text{curl}\,\boldsymbol{\phi}) \times \boldsymbol{v} = (\nabla\boldsymbol{\phi} - \nabla\boldsymbol{\phi}^T)\boldsymbol{v}$$

for every vector $\boldsymbol{v}$. Unlike gradient and divergence, the usual setting of curl can be given only in $\mathbb{E}^3$. A more general definition of curl requires exterior calculus.

Let us remember that (a) a tensor $\boldsymbol{W}$ is said to be *skew* if $\boldsymbol{W} = -\boldsymbol{W}^T$, and (b) there is a one-to-one correspondence between skew tensors and vectors. In fact, for every skew tensor $\boldsymbol{W}$ there is one and only one vector $\boldsymbol{w}$ such that

$$\boldsymbol{w} \times \boldsymbol{v} = \boldsymbol{W}\boldsymbol{v}$$

for every vector $\boldsymbol{v}$. The one-dimensional space spanned by $\boldsymbol{w}$ is called *axis* of the tensor $\boldsymbol{W}$. It coincides with the null space of $\boldsymbol{W}$, i.e. with the subset of vectors that $\boldsymbol{W}$ maps to $\mathbf{0}$.

Clearly enough, $\text{curl}\,\boldsymbol{\phi}$ is the axis of $\nabla\boldsymbol{\phi} - \nabla\boldsymbol{\phi}^T$, which is a skew tensor field.

5.3.4 Gradient, divergence and curl of a 3D field

In elementary vector calculus in $\mathbb{E}^3$ it is customary to use the operator

$$\nabla = \frac{\partial}{\partial x}\boldsymbol{e}_1 + \frac{\partial}{\partial y}\boldsymbol{e}_2 + \frac{\partial}{\partial z}\boldsymbol{e}_3,$$

referred to as either *nabla* or *DEL*, as a sort of vector which follows the standard rules of vector calculus. Using this operator it is possible to give three simple and well-known formulas for gradient, divergence and curl, respectively.

Gradient If ϕ is a differentiable scalar field, then the gradient of ϕ is defined by

$$\text{grad } \phi = \nabla\phi = \frac{\partial\phi}{\partial x}\boldsymbol{e}_1 + \frac{\partial\phi}{\partial y}\boldsymbol{e}_2 + \frac{\partial\phi}{\partial z}\boldsymbol{e}_3$$

Divergence If $\boldsymbol{\phi} = \phi_1\boldsymbol{e}_1 + \phi_2\boldsymbol{e}_2 + \phi_3\boldsymbol{e}_3$ is a differentiable vector field, then the *divergence* of $\boldsymbol{\phi}$ is given by

$$\text{div } \boldsymbol{\phi} = \nabla \cdot \boldsymbol{\phi} = \frac{\partial\phi_1}{\partial x} + \frac{\partial\phi_2}{\partial y} + \frac{\partial\phi_3}{\partial z}$$

Curl If $\boldsymbol{\phi}$ is a differentiable vector field, then the *curl* of $\boldsymbol{\phi}$ is defined by

$$\operatorname{curl} \boldsymbol{\phi} = \nabla \times \boldsymbol{\phi} = \det \begin{pmatrix} \boldsymbol{e}_1 & \boldsymbol{e}_2 & \boldsymbol{e}_3 \\ \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial o}{\partial z} \\ \phi_1 & \phi_2 & \phi_3 \end{pmatrix}$$

$$= \left(\frac{\partial \phi_3}{\partial y} - \frac{\partial \phi_2}{\partial z} \right) \boldsymbol{e}_1 + \left(\frac{\partial \phi_1}{\partial z} - \frac{\partial \phi_3}{\partial x} \right) \boldsymbol{e}_2 + \left(\frac{\partial \phi_2}{\partial x} - \frac{\partial \phi_1}{\partial y} \right) \boldsymbol{e}_3.$$

Laplacian If ϕ is a differentiable scalar field, then the *Laplacian* of ϕ is given by

$$\Delta \phi = \nabla \cdot (\nabla \phi) = \frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \frac{\partial^2 \phi}{\partial z^2} = \nabla^2 \phi$$

where ∇^2 is called the *Laplacian operator*.

Properties Some well-known statements relating gradient, divergence and curl say that, for every differentiable vector field $\boldsymbol{\phi}$ and scalar field ϕ:

1. the curl of the gradient is zero: $(\operatorname{curl} \circ \operatorname{grad})\ \phi = \nabla \times (\nabla \phi) = \mathbf{0}$;
2. a vector field is (locally) a gradient if and only if its curl vanishes;
3. the divergence of curl is zero: $(\operatorname{div} \circ \operatorname{curl})\ \boldsymbol{\phi} = \nabla \cdot (\nabla \times \boldsymbol{\phi}) = 0$;
4. a vector field is (locally) a curl if and only if its divergence vanishes.

Implementation According to its definition, the `Divergence` of a vector field $\boldsymbol{\phi} : \mathbb{R}^n \to \mathbb{R}^n$ is implemented in Script 5.3.4 as the `Trace` of Jacobian matrix of $\boldsymbol{\phi}$. The `Trace` function, that returns the sum of diagonal elements of a square matrix, was introduced in Script 3.3.10.

The `Curl` operator is conversely defined here only for 3D fields, according to the formula $\nabla \times \boldsymbol{\phi}$. Remember that `(S2~S3):M`, where `M` is a `PLaSM` matrix (stored by rows), returns the second element of the third row, i.e. $\mathtt{m}_{32}$.

Script 5.3.4 (Divergence and Curl)

```
DEF Divergence (f::IsSeqOf:IsFun)(x::IsSeqOf:IsReal) =
   Trace:(Jacobian:f:x);

DEF Curl (f::IsSeqOf:IsFun)(x::IsSeqOf:IsReal) =
    [ S2~S3 - S3~S2, S3~S1 - S1~S3, S1~S2 - S2~S1 ]:(Jacobian:f:x)
```

Example 5.3.2 (Checks on field operators)
Some computational checks are done in Script 5.3.5, with the aim of showing both the usage and the result of some compound functional expressions involving curl, divergence and gradient.

The reader should notice that fields are written either as single functions (scalar fields), or as sequences of functions (vector fields), and using the variable-free notation.

In particular, `f1` and `f2` in Script 5.3.5 are two scalar fields in $\mathbb{R}^3$, and `g` is a vector field $\mathbb{R}^3 \to \mathbb{R}^3$.

According to our expectations, we can see that (curl $\circ$ grad) $\phi = \mathbf{0}$ and that (div $\circ$ curl) $\boldsymbol{\phi} = 0$. Those checks are done in a single point, but it is reasonable to expect that they give the same result everywhere, as the reader may check.

Notice also the inefficiency involved in extracting the coordinate functions of the vector functions returned by expressions `Gradient:f1`, `Gradient:f2` and `Curl:g`. At present, there is no way in `PLaSM` to deal efficiently with this kind of extraction.

Script 5.3.5 (Some computational checks)

```
DEF f1 = ( s1*s1 + s2*s2 )/K:2;
DEF f2 = ( s1*s1 + sin ~ s2 - s1*s3 )/ K:2;
DEF g  = < sin ~ s1, cos ~ s2, s1*s3 >;

Gradient:f1:<0.5,0.1> == <0.49999999999994493, 0.10000000000010001>

Curl:< s1 ~ Gradient:f1, s2 ~ Gradient:f1, s3 ~ Gradient:f1 >:<0.9,8,0>
   == <0.0, 0.0, 0.0>
Curl:< s1 ~ Gradient:f2, s2 ~ Gradient:f2, s3 ~ Gradient:f2 >:<1,0.5,1>
   == <0.0, 0.0, 0.0>

Divergence:< s1 ~ Curl:g, s2 ~ Curl:g, s3 ~ Curl:g >:<0.5,110.5,1> == 0.0
Divergence:< s1 ~ Curl:g, s2 ~ Curl:g, s3 ~ Curl:g >:<-8.5,0.5,-3> == 0.0
```

Actually such a solution would exist, but it requires us to completely renounce the λ-style of writing `PLaSM` functions, and its somewhat static typing, and to adopt systematically a *pure* `FL` style, i.e. *without* formal parameters. Doing so, we would be able to write an expression like

```
Divergence:([s1,s2,s3] ~ Curl:g):<0.5,110.5,1>
```

This is not possible with the definition of `Divergence` operator given in Script 5.3.4, because the interpreter would expect to apply it to a *sequence* of functions, whereas `[s1,s2,s3] ~ Curl:g` is a single function.

Actually, there are some drawbacks in using only the pure `FL` style, including less self-documented and more intricate programs, and lack of any type checking. A carefully chosen compromise between λ-style and pure `FL` style is often preferable.

5.4 Differentiable manifolds

In some areas of geometric modeling, and in particular in solid modeling, the terms "manifold" and "differentiable manifold" are used quite frequently. A quick introduction to this concept may therefore be needed, and is given here. A manifold is a set where it makes sense to introduce coordinates, at least locally. Such coordinates locally behave like affine coordinates. Useful references are Crampin and Pirani [CP86], Jones, Gray and Hutton [JGH87], and Nakahara [Nak90].

5.4.1 Charts and atlases

The main concepts here are related to systems of coordinates, and are quite easy to understand. For several sets, for example the surface S_2 of the Earth, only one system of coordinates is not sufficient, because a single bijection between the S_2 points and 2-tuples of coordinates does not exist. Therefore the goal is to introduce coordinate systems on point *subsets* such that:

1. nearby points have nearby coordinates;
2. every point has unique coordinates in each system that contains it;
3. when two coordinate systems overlap, they must be related in a smooth way.

The last requirements guarantee that differentiable functions in one system are differentiable also in the other.

A *chart* for a set M is a pair $(U, \boldsymbol{\phi})$, with $U \subset M$ and $\boldsymbol{\phi}$ a bijective function from U to an open set in $\mathbb{R}^n$. The subset U is called the *coordinate neighborhood*, while $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_n)$ is the *coordinate map*, where $\phi_i : U \to \mathbb{R}$ $(1 \leq i \leq n)$ are the *coordinate functions*.

Two charts $(U, \boldsymbol{\phi})$ and $(V, \boldsymbol{\psi})$ for the set M are said to be *compatible* if, provided that $U \cap V \neq \emptyset$, then:

1. the sets $\boldsymbol{\phi}(U \cap V)$ and $\boldsymbol{\psi}(U \cap V)$ are open subset of $\mathbb{R}^n$;
2. the map $\boldsymbol{\phi} \circ \boldsymbol{\psi}^{-1}$ is smooth with smooth inverse, i.e. is a *diffeomorphism*.

An *atlas* for the set M is a collection $\{(U_i, \phi_i)\}$ of pairwise compatible charts such that $\cup_i U_i = M$.

5.4.2 Differentiable manifolds

Two atlases for the same set M are *equivalent* if every chart in an atlas is compatible with every chart in the other atlas. This relation is reflexive, symmetric and transitive, i.e. is an equivalence relation.

Each equivalence class in the set of atlases of M is said to be a *differentiable structure* for M. A *differentiable manifold* is a pair $(M, \mathcal{S})$, where $\mathcal{S}$ is a differentiable structure for M.

The one-chart atlas $(\mathbb{R}^n,\ \mathrm{id} : \mathbb{R}^n \to \mathbb{R}^n)$ determines a differentiable structure for $\mathbb{R}^n$ as a manifold. This one is probably the most trivial example, where a single chart covers the whole space, and where the coordinate map is the identity map.

Example 5.4.1 (Atlas for the circle S_1)
Let us consider the one-dimensional circle $S_1 = \{\boldsymbol{x} \in \mathbb{E}^2 |\ \|\boldsymbol{x} - \boldsymbol{o}\|^2 = 1\}$, and choose two charts $(U_1, \boldsymbol{\phi}_1)$ and $(U_2, \boldsymbol{\phi}_2)$, with

$$U_1 = S_1 \backslash \{(1, 0)\}, \quad \boldsymbol{\phi}_1 : U_1 \to (0, 2\pi) \subset \mathbb{R} : (\cos\alpha, \sin\alpha) \mapsto \alpha,$$

and

$$U_2 = S_1 \backslash \{(-1, 0)\}, \quad \boldsymbol{\phi}_2 : U_2 \to (-\pi, \pi) \subset \mathbb{R} : (\cos\beta, \sin\beta) \mapsto \beta.$$

It this case it is easy to see that U_1 and U_2 are open sets and that $\boldsymbol{\phi}_1$ and $\boldsymbol{\phi}_2$ are smooth. Then, it can be seen that

$$\boldsymbol{\phi}_1 \circ \boldsymbol{\phi}_2^{-1} = \begin{cases} \text{id} & on \quad (0, \pi) \\ \\ \text{id} - \underline{2\pi} & on \quad (\pi, 2\pi) \end{cases}$$

is a diffeomorphism between $\boldsymbol{\phi}_2(U_1 \cap U_2) = (0, 2\pi) \backslash \{\pi\}$ and $\boldsymbol{\phi}_1(U_1 \cap U_2) = (-\pi, \pi) \backslash \{0\}$. The two charts are hence compatible, and $U_1 \cup U_2 = S_1$. Therefore we can conclude that $\{(U_1, \boldsymbol{\phi}_1), (U_2, \boldsymbol{\phi}_2)\}$ is an atlas for S_1.

Manifold dimension Let M be a differentiable manifold, and let $\mathcal{A} = \{(U_i, \phi_i)\}$ be an atlas for it. If all the sets $\phi_i(U_i)$ are contained in the same $\mathbb{R}^n$, then n is called the *dimension* of the manifold. In other words, the dimension of a differentiable manifold is the (constant) number of coordinates of manifold points in all its charts. This number is the same for any atlas of the manifold.

Not all differentiable manifolds have a dimension, since they may contain subsets covered by charts with different numbers of coordinates.

Product manifold The *product* $M \times N$ of two manifolds M and N of dimension m and n respectively, with atlases $\{(U_i, \phi_i)\}$ and $\{(V_j, \psi_j)\}$, is a manifold of dimension $m + n$, whose atlas is $\{(U_i \times V_j, (\phi_i, \psi_j))\}$.

In this case if $(\boldsymbol{p}, \boldsymbol{q}) \in U_i \times V_j$, then

$$(\phi_i, \psi_j)(\boldsymbol{p}, \boldsymbol{q}) = (\phi_i(\boldsymbol{p}), \psi_j(\boldsymbol{q})). \tag{5.4}$$

Spheres and toruses The n-dimensional sphere S_n is defined as the set $\{\boldsymbol{x} \in \mathbb{E}^{n+1} : \|\boldsymbol{x} - \boldsymbol{o}\|^2 = 1\}$.

The 2-dimensional torus T_2 is the product manifold

$$T_2 = S_1 \times S_1.$$

Clearly it is $T_2 \subset \mathbb{E}^4$, since $S_1 \subset \mathbb{E}^2$. The product manifold called m-dimensional torus

$$T_m = S_1 \times \cdots \times S_1, \quad m \text{ times.}$$

with $T_m \subset \mathbb{E}^{2m}$, is an immediate generalization.

Example 5.4.2 (Torus as Cartesian product of circles)
Our aim in this example is to generate a 3D projection of

$$T_2 = S_1 \times S_1.$$

In Script 5.4.1 we generate the 3D surface with the shape of a doughnut by translating and projecting against the subspace $\{\boldsymbol{x} \in \mathbb{E}^4 | x_4 = 0\}$ the polyhedral complex `torus`, generated by Cartesian product of circle approximations `MAP:(cc:8):d1` and `MAP:(cc:2):d1`, where 8 and 2 are the radiuses, and `d1` is a

Script 5.4.1 (Torus as product of circles (1))

```
DEF cc (r::IsRealPos) = [ K:r * COS, K:r * SIN ] ~ S1;
DEF d1 = interval:< 0,2*PI >:24
DEF torus = (MAP: (cc:8): d1) * (MAP: (cc:2): d1);

VRML:((schlegel3D:2 ~ T:4:-11):torus CREASE (PI/2)):'out2.wrl' ;
```

partition of $[0, 2\pi]$ with 24 segments. The `schlegel3D` operator is given and discussed in Section 4.6.1. The `interval:` function is given in Script 5.1.1.

Notice that both circles, with $r = 8$ and $r = 2$ respectively, were produced by `MAP` operator as $\boldsymbol{\phi}^{-1}[0, 2\pi]$, with a single chart $(U, \boldsymbol{\phi})$ depending on r, where

$$U = \{\boldsymbol{x} \in \mathbb{E}^2 | (\boldsymbol{x} - \boldsymbol{o})^2 = r^2\}, \quad \boldsymbol{\phi} : U \to [0, 2\pi] : (r\cos\alpha, r\sin\alpha) \mapsto \alpha,$$

thus producing a singularity (a double coordinate value which results in the same point) in $\boldsymbol{\phi}^{-1}(0) = (r, 0)$ and in $\boldsymbol{\phi}^{-1}(2\pi) = (r, 0)$. This kind of singularity, i.e. double points at the boundary of charts of closed surfaces, will be accepted through the whole book, for the sake of implementation simplicity, in particular because the `PLaSM` kernel is currently able to manage only closed (i.e. with boundary) polyhedral complexes.

Example 5.4.3 (Torus as manifold product of circles)
Given the charts $(U_1, \boldsymbol{\phi}_1)$ and $(U_2, \boldsymbol{\phi}_2)$ of two circles $S_1(r_1)$ and $S_2(r_2)$ with different radiuses, we generate the torus

$$T_2(r_1, r_2) = S_1(r_1) \times S_1(r_2),$$

using the product chart $(U_1 \times U_2, (\boldsymbol{\phi}_1, \boldsymbol{\phi}_2))$.

A direct implementation of `torus` as the polyhedral approximation of a *product manifold* is hence given in Script 5.4.2, where `c1c2` implements the mapping $(\boldsymbol{\phi}_1, \boldsymbol{\phi}_2)^{-1}$ on $[0, 2\pi]^2$ using the *product chart* $(U_1 \times U_2, (\boldsymbol{\phi}_1, \boldsymbol{\phi}_2))$ described in equation (5.4). In particular, $(\boldsymbol{\phi}_1, \boldsymbol{\phi}_2)^{-1}$ is implemented by the `c1c2` function, whereas a cell decomposition of $[0, 2\pi]^2$ is given by the polyhedral complex `d1 * d1`, where `d1` is given in Script 5.4.1. The simplicial decomposition needed to take into account the double curvature of the doughnut surface is automatically provided by the `MAP` operator.

Script 5.4.2 (Torus as product of circles (2))

```
DEF c1c2 (r1,r2::IsRealPos) =
    [ K:r1 * COS~S1, K:r1 * SIN~S1, K:r2 * COS~S2, K:r2 * SIN~S2 ];
DEF torus (r1,r2::IsRealPos) = MAP: (c1c2:< 8,2 >): (d1 * d1);

VRML:((schlegel3D:2 ~ T:4:-11):(torus:<8,2>) CREASE (PI/2)):'out2.wrl' ;
```

The reader may try unbelievable variations of the generated 3D object by making small variations to the parameters of the expression, including the circle radiuses, the distance of the center of projection from the origin, the translation and the rotation parameters. For example, Figures 5.10a and 5.10c are respectively generated as

```
DEF rotations = (COMP ~ [R:<1,4>,R:<2,4>,R:<3,4>]):(PI/6);
VRML:((project:1):man CREASE (PI/2)):'out1.wrl' ;
VRML:((project:1 ~ rotations):man CREASE (PI/2)):'out3.wrl' ,
```

Figure 5.10 Different projections on $x_4 = 0$ of the manifold $T_2 = S_1 \times S_1 \subset \mathbb{E}^4$: (a) standard projection gives a cylinder (b) central projection (c) multiple rotations followed by projection produce a double self-intersection in the $\mathbb{E}^3$ embedding

Finally, let us notice that parametric equations of T_2 in $\mathbb{E}^4$

$$x_1 = r_1 \cos u, \quad x_2 = r_1 \sin u, \quad x_3 = r_2 \cos v, \quad x_4 = r_2 \sin v,$$

where $(u, v) \in [0, 2\pi]^2$, are simpler than equations (2.5) in $\mathbb{E}^3$.

5.4.3 Tangent spaces and maps

A *tangent vector* to M at x is an element of $\boldsymbol{T}_x\mathbb{R}^n$ of the form $(x, D\gamma(0)(1))$, where $x \in M$ and $\gamma : I \to M$ is a smooth parametrized curve with $\gamma(0) = x$.

The **tangent space** to M at x is the set of all tangent vectors to M at x. It is denoted by $\boldsymbol{T}_x M$.

The tangent bundle of M is the union of all its tangent spaces:

$$\boldsymbol{T}M = \bigcup_{x \in M} \boldsymbol{T}_x M.$$

Let $f : M \to N$, with M and N submanifolds of $\mathbb{R}^m$ and $\mathbb{R}^n$, and $x \in M$. A tangent map will send a small piece of curve through x to a small piece of curve through $f(x)$.

The *tangent map* of f at x is the map:

$$\boldsymbol{T}_x f : \boldsymbol{T}_x M \to \boldsymbol{T}_{f(x)} N$$

such that

$$(x, D\gamma(0)(1)) \mapsto (f(x), D(f \circ \gamma)(0)(1))$$

for each differentiable curve γ through x. The **tangent map** of f is the map

$$\boldsymbol{T}f : \boldsymbol{T}M \to \boldsymbol{T}N : \boldsymbol{T}_x M = \boldsymbol{T}_x f, \quad \text{for each } x \in M.$$

5.5 Derivatives of a curve

An intrinsic representation of a curve, in the sense that it depends only on operations carried out on the curve itself and does not rely on the ambient Euclidean space, is

obtained by repeated application of the derivation operator to the curve-generating function. It is only necessary that this function is (at least) twice differentiable for plane curves, and (at least) three times differentiable for space curves. Let us assume only smooth curves. This case is certainly the more frequent and useful one in CAD applications.

Tangent The *tangent* $\boldsymbol{t}$ vector field, also denoted as $\boldsymbol{c}'$, to a curve $\boldsymbol{c} : \mathbb{R} \to \mathbb{E}^d$, with $\boldsymbol{c} = (c_i)$, is defined as the vector function $\boldsymbol{c}' : \mathbb{R} \to \mathbb{R}^d$ such that:

$$\boldsymbol{t} = D\boldsymbol{c} = \boldsymbol{c}' = (c_i') \, .$$

For each u parameter value, $\boldsymbol{t}(u)$ gives the tangent vector to the curve at $\boldsymbol{c}(u)$.

If $u = t$ is the time t, then $\boldsymbol{v}(t) = D\boldsymbol{c}(t)$ represents the *velocity* with which the point $\boldsymbol{c}(t)$ describes the curve image. Analogously, $\boldsymbol{a}(t) = D\boldsymbol{v}(t) = D^{(2)}\boldsymbol{c}(t)$ represents the *acceleration*.

If $u = s$ is the curve length from $\boldsymbol{c}(0)$, then $\boldsymbol{t}(s)$ gives the unit tangent vector at curvilinear abscissa s. It is in fact easy to see that $\|\boldsymbol{t}(s)\| = 1$ for every s.

Normal Analogously, a *normal* to a twice derivable curve $\boldsymbol{c} : \mathbb{R} \to \mathbb{E}^d$ is defined as the vector function field: $\boldsymbol{c}^{(2)} : \mathbb{R} \to \mathbb{R}^d$ such that:

$$\boldsymbol{c}^{(2)} = D^{(2)}\boldsymbol{c} = (c_i^{(2)}).$$

For each u parameter value, $\boldsymbol{c}^{(2)}(u)$ gives one of normal vectors to the tangent vector for the corresponding point $\boldsymbol{c}(u)$ of the curve.

It is important to notice that in Euclidean space a vector function $\boldsymbol{v}(t)$ of constant norm has the property that $\boldsymbol{v}(t)$ and $D\boldsymbol{v}(t)$ are orthogonal.

Curvature The *curvature* κ of a twice differentiable curve $\boldsymbol{c} : \mathbb{R} \to \mathbb{E}^d : s \mapsto \boldsymbol{o} + \boldsymbol{\alpha}(s)$, where s is the curve length from $\boldsymbol{c}(0)$, is defined as the norm of the vector function $D\boldsymbol{t}$:

$$\kappa : \mathbb{R} \to \mathbb{R} : s \mapsto \|D\boldsymbol{t}(s)\| = \left\| D^{(2)}\boldsymbol{\alpha}(s) \right\|.$$

In other words, the curvature $\kappa(s)$ is the magnitude of the acceleration vector to the curve $\boldsymbol{c}(s)$, under the condition that this one is parametrized by the arc length s.

Principal normal If $D\boldsymbol{t}(s) \neq \boldsymbol{0}$ for all s, then the unit vector function $\boldsymbol{n} : \mathbb{R} \to \mathbb{R}^d$, such that

$$\boldsymbol{n} = \frac{D\boldsymbol{t}}{\|D\boldsymbol{t}\|} = \kappa^{-1} D\boldsymbol{t},$$

is called *principal normal* vector field.

Binormal The plane which contains both $\boldsymbol{t}$ and $\boldsymbol{n}$ at u is called the *osculating plane* at u. The word comes from Latin, for the plane which contains the osculating circle, i.e. the circle "kissing" the curve. The vector product of tangent and normal vector is called the *binormal* vector field:

$$\boldsymbol{b} = \boldsymbol{t} \times \boldsymbol{n}$$

The norm of $D\boldsymbol{b}$ gives the rate of variation of the osculating plane along the curve. If $u = s$ is the arc-length of the curve $\boldsymbol{c}$, then it is possible to write

$$D\boldsymbol{b} = -\tau \boldsymbol{n}$$

where the function $\tau = \tau(s)$ is called *torsion* at $\boldsymbol{c}(s)$.

The *fundamental theorem* of differential geometry of curves in $\mathbb{E}^3$ says that a curve is completely determined by its curvature and torsion functions within a congruence.

The Serret-Frenet formulæ Let us study a space curve $\boldsymbol{c}$ parametrized by the arc length s, also called the *natural parameter*.

The tangent $\boldsymbol{t}$, the principal normal $\boldsymbol{n}$ and the binormal $\boldsymbol{b}$ vector functions are related to their derivatives by the curvature κ and the torsion τ, as shown by the Serret-Frenet formulæ:

$$D\begin{pmatrix} \boldsymbol{t} \\ \boldsymbol{n} \\ \boldsymbol{b} \end{pmatrix} = \begin{pmatrix} 0 & \kappa & 0 \\ -\kappa & 0 & \tau \\ 0 & -\tau & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{t} \\ \boldsymbol{n} \\ \boldsymbol{b} \end{pmatrix}$$

Center of curvature For an arbitrary curve $\boldsymbol{a} \in \mathbb{E}^n$, the curvature field $\kappa : \mathbb{R} \to \mathbb{R}$ is not constant. Where $\kappa(s) \neq 0$, the curve is approximated by a circle of radius $1/\kappa(s)$, which is tangent to $\boldsymbol{a}$ in $\boldsymbol{a}(s)$ and which is contained in the plane generated by $\boldsymbol{t}(s)$ and $\boldsymbol{n}(s)$.

This circle, called *osculating circle*, has the same tangent, curvature and principal normal of the curve in s. The center of the osculating circle is called *center of curvature* of $\boldsymbol{a}$ in $\boldsymbol{a}(s)$. It is a field $\boldsymbol{c} : \mathbb{R} \to \mathbb{R}^d$ that can be written, using the variable-free notation, as:

$$\boldsymbol{c} = \boldsymbol{a} + \kappa^{-1}\boldsymbol{n}.$$

5.6 Examples

Helix The *circular helix* curve in $\mathbb{E}^3$ has vector equation $\boldsymbol{s} = \boldsymbol{o} + \boldsymbol{\beta}(u)$, where the u parameter denotes the angle between the $\boldsymbol{s}$ projection in $z = 0$ and the x axis. For a helix segment we have $\boldsymbol{\beta} : U \to \mathbb{R}^3$, with $U \subset \mathbb{R}$, and

$$\boldsymbol{\beta}(u) = \begin{pmatrix} a\cos u & a\sin u & bu \end{pmatrix} \tag{5.5}$$

where a is the radius of the circular projection in $z = 0$ and b is the ratio $pitch/2\pi$. The *pitch* of the helix is the vertical distance of two curve points differing for 2π in the parameter value.

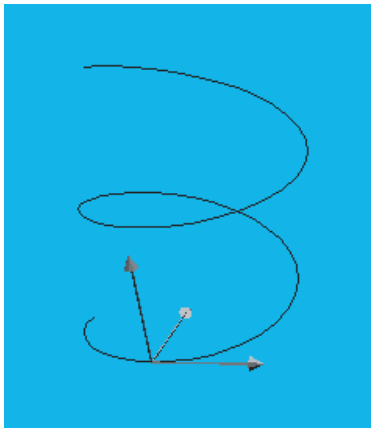

Figure 5.11 Intrinsic triplet at a helix point

Helix curvature and principal normal In Script 5.6.1 we have implemented a `curvature` and `principalNormal` functions, by abstracting with respect to the argument curve. The geometric object generated by evaluating the `STRUCT` expression is shown in Figure 5.11. The `helix` curve `domain` is defined in Script 5.2.7. The `MKvector` function is given in Script 6.5.2.

Notice that, according to the `STRUCT` semantics discussed in Chapter 8, the translation operator is applied to all subsequent vector models. The intrinsic triplet there generated is associated to the value $\frac{\pi}{2}$ of the curve parameter.

Script 5.6.1 (Intrinsic triplet)

```
DEF tangent (curve::IsSeqOf:IsFun) = UnitVect ~ CONS:(AA:D:curve);
DEF curvature (curve::IsSeqOf:IsFun) = VectNorm ~ CONS:(AA:(D ~ D):curve);
DEF principalNormal (curve::IsSeqOf:IsFun) = CONS:
   ((K:1 / curvature:curve) scalarVectProd (AA:(D ~ D):curve));
DEF binormal (curve::IsSeqOf:IsFun) =
   tangent:curve LIFT:vectProd principalNormal:curve ;

DEF curve = helix:<1,0.2>;
STRUCT:<
   MAP:(CONS:curve ~ s1):(interval:<0,4*PI>:60),
   T:<1,2,3>:(CONS:curve:(PI/2)),
   MKvector:<0,0,0>:(tangent:curve:(PI/2)),
   MKvector:<0,0,0>:(principalNormal:curve:(PI/2)),
   MKvector:<0,0,0>:(binormal:curve:(PI/2))
>;
```

Helix parametrized by arc length Let us reparametrize the helix (5.5) as a function of the arc length s. So we have:

$$D\boldsymbol{\beta}(u) = \begin{pmatrix} -a\sin u & a\cos u & b \end{pmatrix}$$

$$\frac{ds}{du} = \|D\boldsymbol{\beta}(u)\| = \sqrt{a^2+b^2}$$

$$s = \int_0^s ds = \int_0^u \sqrt{a^2+b^2}\; du = u\sqrt{a^2+b^2}$$

and hence we can write $\boldsymbol{s} = \boldsymbol{o} + \boldsymbol{\alpha}(s)$, with $s \in [0, L]$ and

$$\boldsymbol{\alpha}(s) = \begin{pmatrix} a\cos\frac{s}{\sqrt{a^2+b^2}} & a\sin\frac{s}{\sqrt{a^2+b^2}} & b\frac{s}{\sqrt{a^2+b^2}} \end{pmatrix}$$

Two helixes of the same length $L = 6\pi$ generated by $\boldsymbol{\alpha}(s)$ are shown in Figure 5.12.

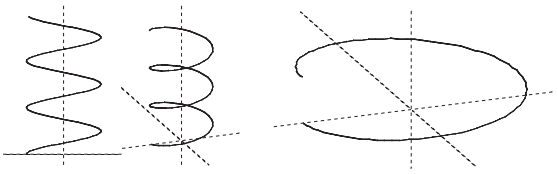

Figure 5.12 Two helixes of the same length and different radius

Osculating circle to the helix Our aim here is to produce the geometric construction shown in Figures 5.13 and 5.14, where a helix curve is displayed together with the curve of its centers of curvature (also a helix) and with the osculating circle at a point.

So, we start by computing the tangent vector field

$$\boldsymbol{t}(s) = D\boldsymbol{\alpha}(s) = \frac{1}{\sqrt{a^2+b^2}}\begin{pmatrix} -a\sin\frac{s}{\sqrt{a^2+b^2}} & a\cos\frac{s}{\sqrt{a^2+b^2}} & b \end{pmatrix}$$

which is a unit vector, and the second derivative

$$D\boldsymbol{t}(s) = \kappa(s)\,\boldsymbol{n}(s) = \frac{1}{a^2+b^2}\begin{pmatrix} -a\cos\frac{s}{\sqrt{a^2+b^2}} & -a\sin\frac{s}{\sqrt{a^2+b^2}} & 0 \end{pmatrix}$$

so that we get

$$\kappa(s) = \frac{a}{a^2+b^2}$$

$$\boldsymbol{n}(s) = \begin{pmatrix} -\cos\frac{s}{\sqrt{a^2+b^2}} & -\sin\frac{s}{\sqrt{a^2+b^2}} & 0 \end{pmatrix}$$

Let us note that the helix is a curve of constant curvature.

If $a \neq 0$ and $b = 0$, then the curve is a circle in xy plane and $\kappa = 1/a$. In other words, the curvature is the reciprocal of radius.

If $a = 0$ and $b \neq 0$, then the curve is a segment of straigth line, with $\kappa(s) = 0$ and $\boldsymbol{n}(s)$ undefined.

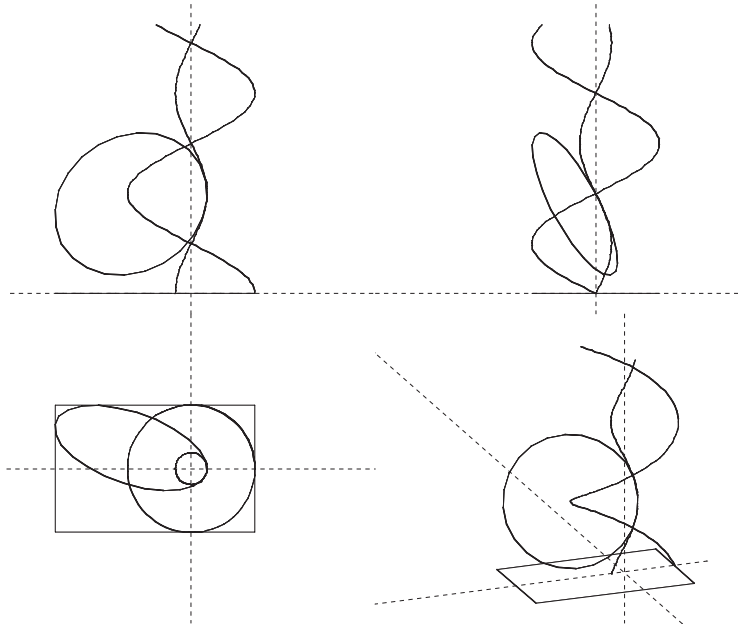

Figure 5.13 Views of an "artistic" assembly including a helix, the locus of the centers of osculating circles and the osculating circle at a point

Implementation We start by implementing, in Script 5.6.2, the `helix` vector function $\boldsymbol{\alpha} : \mathbb{R} \to \mathbb{R}^3$ and the intrinsic triplet of orthonormal vector functions $\boldsymbol{t}$, $\boldsymbol{n}$ and $\boldsymbol{b}$. As always, we give a direct translation of the variable-free formulation of the $\boldsymbol{\alpha}$ function, with

$$\boldsymbol{\alpha} = \left(\underline{a}\cos\circ\frac{\mathrm{id}}{\mathrm{id}^{\frac{1}{2}}\circ\underline{a^2+b^2}} \quad \underline{a}\sin\circ\frac{\mathrm{id}}{\mathrm{id}^{\frac{1}{2}}\circ\underline{a^2+b^2}} \quad \underline{b}\,\frac{\mathrm{id}}{\mathrm{id}^{\frac{1}{2}}\circ\underline{a^2+b^2}} \right)$$

Notice that, for the sake of abstraction and consistency with the contents of this chapter, we have used the derivation operator `D` in the definition of the `tangent` and `normal` functions, instead using the symbolic expressions previously given.

Script 5.6.2 (Helix parametrized by arc length)

```
DEF helix (a,b::IsReal) = <K:a * cos ∼ u, K:a * sin ∼ u, K:b * u>
WHERE
   u = ID / SQRT ∼ K:(a*a + b*b)
END;

DEF helixtangent (a,b::IsReal) = AA:D:(helix:<a,b>);
DEF helixnormal (a,b::IsReal) =
   K:((a*a + b*b)/a) scalarVectProd AA:(D ∼ D):(helix:<a,b>);
DEF helixbinormal (a,b::IsReal) = helixtangent:<a,b> VectProd helixnormal:<a,b>;
```

Notice also that in Script 5.6.2 we have redefined some function identifier already used with a more general meaning. So, in the remainder of this chapter the functions

called `tangent`, `normal` and `binormal` uniquely refer to the unit vector triplet intrinsically related to the `helix` curve parametrized by the arc length.

In Script 5.6.3 it is first of all defined the `CurvatureCenter` $\mathbb{R} \to \mathbb{R}^3$ curve, generated as the vector sum of the `helix` $\mathbb{R} \to \mathbb{R}^3$ curve plus the `normal` $\mathbb{R} \to \mathbb{R}^3$ curve times the `K:((sqr:a + sqr:b)/a)` function $\mathbb{R} \to \mathbb{R}$. Actually the implementation produces a whole family of such curves, parametrized by the pair of real parameters a, b, corresponding respectively to the helix radius and to the z interval between two helix points at unit curvilinear distance.

In the same script the `OsculCircle` function is also given, which generates the osculating circle at curvilinear ascissa s for the curve `helix:<a,b>`. The image of such function in a concrete case is shown in Figure 5.13.

Script 5.6.3 (Osculating circle)

```
DEF helixcurvatureCenter (a,b::IsReal) =
   helix:<a,b> vectSum (K:((sqr:a + sqr:b)/a) scalarVectProd helixnormal:<a,b>)

DEF OsculCircle (a,b,s::IsReal) =
   (T:<1,2,3>:center ∼ Rotn:<angle, axis>):circle
WHERE
   circle = MAP:((circle3D ∼ radius):<a,b, s>):(interval:<0,2*PI>:24),
   angle = (- ∼ ACOS ∼ InnerProd):<ortho,<0,0,1>>,
   axis = CONS:(helixnormal:<a,b>):s,
   center = CONS:(helixCurvatureCenter:<a,b>):s,
   ortho  = CONS:(helixbinormal:<a,b>):s
END;
```

Some utility functions needed to specify the geometric assembly of Figure 5.13 are given in Script 5.6.4. These include:

1. a function `circle3D` to produce a circle of radius `r` embedded in the coordinate subspace $z = 0$ and centered at the origin;
2. a function `radius` to compute the numeric value of the radius of the osculating circle at curvilinear abscissa s. Such a number is computed as the norm of the vector difference of two corresponding points on the `helix:<a,b>` and `curvatureCenter:<a,b>` curves;
3. a function `CurveGraph` to generate a simplicial approximation with 90 line segments of the image `f`$[0, 6\pi]$ of its vector function argument `f`;

Script 5.6.4 (Utility functions)

```
DEF circle3D (r::IsReal) = [K:r * COS, K:r * SIN, K:0] ∼ s1;
DEF radius (a,b,s::IsReal) = (VectNorm ∼ VectDiff):
   < CONS:(helix:<a,b>):s, CONS:(CurvatureCenter:<a,b>):s >;
DEF CurveGraph (f::IsSeqOf:IsFun) = MAP:(CONS:f ∼ s1):(interval:<0,6*PI>:90);
```

Finally, in Script 5.6.5 we give the definition of an `assembly` object, whose geometric value is shown in Figure 5.14. Such an "artistic" assembly contains simplicial

approximations of the sets `helix:<1,2>`$[0, 2\pi]$ and `curvatureCenter:<1,2>`$[0, 2\pi]$ and a "mat" (i.e. 2D) version of the osculating circle at $s = 2\pi$, as produced by the `JOIN` primitive.

Script 5.6.5 (Artistic assembly)

```
DEF assembly = STRUCT:<
   (CurveGraph ~ helix):<1,2>,
   (CurveGraph ~ curvatureCenter):<1,2>,
   (JOIN ~ OsculCircle):<1,2, 2*PI>
>;

(STRUCT ~ [EMBED:1 ~ BOX:<1,2>, ID]):assembly;
```

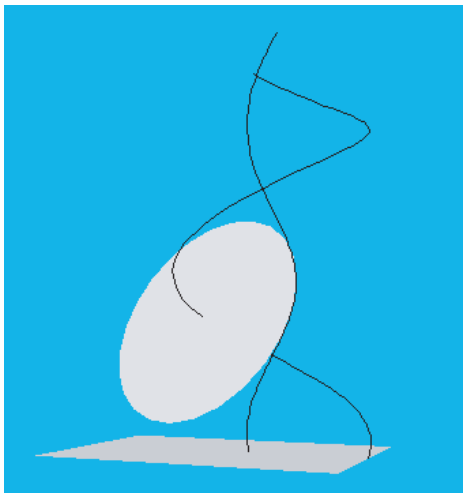

Figure 5.14 "Artistic" assembly of: (i) a `helix` curve (ii) the locus of the centers of its osculating circles (iii) the osculating circle at $s = 2\pi$

5.7 Intrinsic properties of a surface

According to the definition of curves, a surface can be defined as a *point-valued* function $\boldsymbol{s}$ of *two* real variables. So, a *surface* in $\mathbb{E}^d$ is generated as a set of points by summing to the origin a vector-valued function $\boldsymbol{\beta} : \mathbb{R}^2 \to \mathbb{R}^d$ of two real variables:

$$\boldsymbol{s}(u, v) = \boldsymbol{o} + \boldsymbol{\beta}(u, v), \qquad (u, v) \in [0, 1]^2 \subset \mathbb{R}^2.$$

The *image* of the surface is the set $\boldsymbol{s}[0, 1]^2$ of its $\mathbb{E}^d$ points. The *domain* of the surface is the parameter interval $[0, 1]^2$.

A surface is said to be *regular* where the partial derivatives

$$\boldsymbol{s}_1(u, v) = \frac{\partial \boldsymbol{s}}{\partial u}(u, v) \quad \text{and} \quad \boldsymbol{s}_2(u, v) = \frac{\partial \boldsymbol{s}}{\partial u}(u, v)$$

are linearly independent. In the same way, it is regular (in $\mathbb{E}^3$) where

$$\boldsymbol{s}_1(u,v) \times \boldsymbol{s}_2(u,v) \neq \mathbf{0}.$$

In this case, $\boldsymbol{s}_1(u,v)$ and $\boldsymbol{s}_2(u,v)$ give a basis for the tangent space at $\boldsymbol{s}(u,v)$.

Hence the tangent vector $\boldsymbol{c}'(t)$ to a curve $\boldsymbol{c}(t) = \boldsymbol{s}(u(t), v(t))$ on the surface $\boldsymbol{s}$ can be expressed as a linear combination of the partial derivatives of $\boldsymbol{s}$:

$$\boldsymbol{c}' = \frac{du}{dt}\frac{\partial \boldsymbol{s}}{\partial u} + \frac{dv}{dt}\frac{\partial \boldsymbol{s}}{\partial v} = u'\boldsymbol{s}_1 + v'\boldsymbol{s}_2.$$

Normal to a 3D surface The *normal* field $\boldsymbol{n} : \mathbb{R}^2 \to \mathbb{R}^3$ to a surface $\boldsymbol{s} : \mathbb{R}^2 \to \mathbb{E}^3$ is given, where the surface is regular, by the normalized vector product of the partial derivatives:

$$\boldsymbol{n} = \frac{\boldsymbol{s}_1 \times \boldsymbol{s}_2}{\|\boldsymbol{s}_1 \times \boldsymbol{s}_2\|}$$

Implementation In Script 5.7.1 we give the function `DS` that, when applied to an integer `i` and to a surface map `surf`, defined as a *sequence* of coordinate functions $\mathbb{R}^2 \to \mathbb{R}$, returns the partial derivative field $\texttt{surf}_i : \mathbb{R}^2 \to \mathbb{E}^3$ $(1 \leq i \leq 2)$. This result is obtained by suitably applying to each component of `surfMap` the `Dp` linear operator given in Script 5.2.8.

Also, a *normal field* operator `N` is defined, as the normalized vector product of the (tangent) fields generators `DS:1` and `DS:2`. The `vectorProd` operator can be found in Script 3.1.3. Clearly, specific tangent or normal fields are returned when such operators are applied to a specific surface mapping.

Script 5.7.1 (Partial derivatives and normal)

```
DEF X(i::isIntPos)(f::IsFun)(a::IsPoint) = Dp:i:f:a:<1>;
DEF Norm3 (x,y,z::IsFun) = < x/den, y/den, z/den >
WHERE
   den = SQRT ∼ + ∼ AA:sqr ∼ [x, y, z],
   sqr = ID * ID
END;

DEF DS (i::IsIntPos)(surfMap::IsSeqOf:IsFun) = AA:(X:i): surfMap;
DEF N = Norm3 ∼ VectProd ∼ [DS:1, DS:2];
```

Example 5.7.1 (Graph of partial derivatives)
We give in Script 5.7.2 an implementation of the geometric construction shown in Figure 5.15, where a graph is shown of both the tangent vectors $\texttt{DS:1}(\alpha)(\frac{\pi}{5}, \frac{\pi}{10})$ and $\texttt{DS:2}(\alpha)(\frac{\pi}{5}, \frac{\pi}{10})$, and of the normal vector $\texttt{N}(\alpha)(\frac{\pi}{5}, \frac{\pi}{10})$, for a surface $\alpha = (u, v, \sin u \sin v)$. in Figure 5.15 the two curves $\alpha(u, \frac{\pi}{10})$ and $\alpha(\frac{\pi}{5}, v)$ are also shown.

This geometric construction is produced by exporting the geometric value of the `graph` object in Script 5.7.2, for example as:

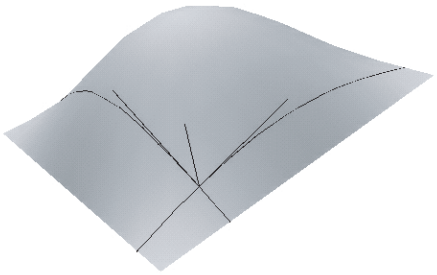

Figure 5.15 First partial derivatives and normal vector in $\alpha(\frac{\pi}{5}, \frac{\pi}{10})$ to a surface $\alpha(u, v) = (u, v, \sin u \sin v)$

```
VRML:graph:'out.wrl';
```

Notice that the variable-free representation (see Section 5.1.1) of the studied surface is

$$\alpha := (\sigma(1), \sigma(2), (\sin \circ \sigma(1))(\sin \circ \sigma(2)))$$

which is directly translated into the `surf` sequence. The functions `interval` and `interval2D`, generating suitable decompositions of a 1D and 2D interval, respectively, are given in Script 5.1.1.

Script 5.7.2 (Graph of partial derivatives)

```
DEF surf = < S1, S2, SIN ~ S1 * SIN ~ S2 >;
DEF domain = interval2D:<0,0,PI,PI>:<10,10>;

DEF graph = STRUCT:<
   MAP:surf:domain CREASE (PI/2),
   MAP:surf:((T:2:(PI/10) ~ EMBED:1 ~ interval:<0,PI>):10),
   MAP:surf:((T:1:(PI/5) ~ R:<1,2>:(PI/2) ~ EMBED:1 ~ interval:<0,PI>):10),
   T:<1,2,3>:p,
   Segment:1:< <0,0,0>, CONS:(DS:1:surf):<PI/5, PI/10> >,
   Segment:1:< <0,0,0>, CONS:(DS:2:surf):<PI/5, PI/10> >,
   Segment:1:< <0,0,0>, CONS:(N:surf):<PI/5, PI/10> >
>
WHERE p = CONS:surf:<PI/5, PI/10> END;
```

Example 5.7.2 (Sampling of normal field)
A graphical representation of the normal field $\boldsymbol{n}$ is produced by the `N_Map` function of Script 5.7.3, and displayed in Figure 5.16 for the normal field induced by the $\alpha(u, v) = (u, v, \sin u \sin v)$ surface, with $(u, v) \in [0, \pi]^2$.

In particular, Figure 5.16 is generated by graphically browsing the geometric value produced by the `PLaSM` interpreter when evaluating the last expression of Script 5.7.3.

The reader should notice that the `N_Map` function given here is very similar to the `GradMap` function of Script 5.3.2.

Script 5.7.3 (Sampling of normal field)

```
DEF N_Map (f::IsSeqOf:IsFun; dom::IsPol) = (STRUCT
   ~ AA:(Segment:0.35 ~ [CONS:f, vectSum ~ [CONS:f,(CONS ~ N):f] ])
   ~ S1 ~ UKPOL):dom;

N_Map:< surf, domain > STRUCT (MAP:surf:domain CREASE (PI/2))
```

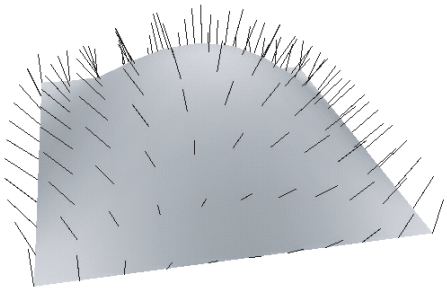

Figure 5.16 Sampling of the (scaled) normal field $\boldsymbol{n}(u,v)$ to a surface $\alpha(u,v) = (u, v, \sin u \sin v)$, over the domain $[0,\pi]^2$

5.7.1 First fundamental form

Let us consider a curve $\boldsymbol{c}(t) = \boldsymbol{s}(u(t), v(t))$ on the surface $\boldsymbol{s}(u,v)$, with $a \le t \le b$, and let s be the arc length:

$$s(t) = \int_a^t ds = \int_a^t \|\boldsymbol{c}'(t)\| dt.$$

Hence we have:

$$\begin{aligned}\left(\frac{ds}{dt}\right)^2 &= \|\boldsymbol{c}'\|^2 = \boldsymbol{c}' \cdot \boldsymbol{c}' = (u'\boldsymbol{s}_1 + v'\boldsymbol{s}_2) \cdot (u'\boldsymbol{s}_1 + v'\boldsymbol{s}_2) \\ &= (\boldsymbol{s}_1 \cdot \boldsymbol{s}_1)u' + 2(\boldsymbol{s}_1 \cdot \boldsymbol{s}_2)u'v' + (\boldsymbol{s}_2 \cdot \boldsymbol{s}_2)v'\end{aligned}$$

from where, by using the Gauss notation:

$$\boldsymbol{s}_1 \cdot \boldsymbol{s}_1 = \mathrm{E}, \qquad \boldsymbol{s}_1 \cdot \boldsymbol{s}_2 = \mathrm{F}, \qquad \boldsymbol{s}_2 \cdot \boldsymbol{s}_2 = \mathrm{G}. \tag{5.6}$$

we have

$$\left(\frac{ds}{dt}\right)^2 = \mathrm{E}u' + 2\mathrm{F}u'v' + \mathrm{G}v',$$

and remembering that $u' = \frac{du}{dt}$ and $v' = \frac{dv}{dt}$, we get the intrinsic quantity called *First Fundamental Form* or also *Metric Form*

$$ds = \mathrm{E}\, du + 2\,\mathrm{F}\, du\, dv + \mathrm{G}\, dv,$$

in differential notation, which arises in computing the arc length of curves on surfaces, as well as the angle between tangent vectors on a surface.

In fact, let $\boldsymbol{v} = a\boldsymbol{s}_1 + b\boldsymbol{s}_2$ and $\boldsymbol{w} = c\boldsymbol{s}_1 + d\boldsymbol{s}_2$ be tangent vectors to a surface $\boldsymbol{s}$ in the same point. We can write

$$\begin{aligned}
\boldsymbol{v}\cdot\boldsymbol{w} &= (a\boldsymbol{s}_1 + b\boldsymbol{s}_2)\cdot(c\boldsymbol{s}_1 + d\boldsymbol{s}_2)\\
&= ac\,(\boldsymbol{s}_1\cdot\boldsymbol{s}_1) + (ad+bc)\,(\boldsymbol{s}_1\cdot\boldsymbol{s}_2) + bd\,(\boldsymbol{s}_2\cdot\boldsymbol{s}_2)\\
&= ac\,\mathrm{E} + (ad+bc)\,\mathrm{F} + bd\,\mathrm{G}\\
&= \begin{pmatrix} a & b \end{pmatrix}\begin{pmatrix} \mathrm{E} & \mathrm{F}\\ \mathrm{F} & \mathrm{G}\end{pmatrix}\begin{pmatrix} c\\ d\end{pmatrix}.
\end{aligned}$$

The matrix in the above formula is called *matrix of the first fundamental form* or also *metric tensor*.

Relabeling $\boldsymbol{s}_i\cdot\boldsymbol{s}_j = g_{ij}$, it is possible to write for the differential area element:

$$\begin{pmatrix} E & F\\ F & G\end{pmatrix} = \begin{pmatrix} g_{11} & g_{12}\\ g_{21} & g_{22}\end{pmatrix}$$

and, by labeling $du = du^1$ and $dv = dv^2$

$$ds^2 = \sum_{i,j} g_{ij}\,du^i\,du^j, \qquad i,j\in\{1,2\}.$$

Another important property of the metric tensor is:

$$\begin{aligned}
\|\boldsymbol{s}_1\times\boldsymbol{s}_2\|^2 &= (\boldsymbol{s}_1\cdot\boldsymbol{s}_1)(\boldsymbol{s}_2\cdot\boldsymbol{s}_2) - (\boldsymbol{s}_1\cdot\boldsymbol{s}_2)^2\\
&= \mathrm{E}\,\mathrm{G} - \mathrm{F}^2\\
&= \det\begin{pmatrix} g_{11} & g_{12}\\ g_{21} & g_{22}\end{pmatrix}\\
&=: g
\end{aligned}$$

Example 5.7.3
Let us try the forms E, F and G on the $(0,0)$ corner of the $[0,1]^2$ domain of the $\boldsymbol{s} = (u, v, \sin u\,\sin v)$ surface map $[0,1]^2\to\mathbb{E}^3$ given in Script 5.7.2.

In this case we have

$$\begin{aligned}
\boldsymbol{s}_1(u,v) &= (1, 0, \cos u\,\sin v)\\
\boldsymbol{s}_2(u,v) &= (0, 1, \sin u\,\cos v),
\end{aligned}$$

so that

$$\begin{aligned}
E(\boldsymbol{s})(0,0) &= (\boldsymbol{s}_1\cdot\boldsymbol{s}_1)(0,0) = ((1,0,\cos u\,\sin v)\cdot(1,0,\cos u\,\sin v))(0,0) = 1\\
F(\boldsymbol{s})(0,0) &= (\boldsymbol{s}_1\cdot\boldsymbol{s}_2)(0,0) = ((1,0,\cos u\,\sin v)\cdot(0,1,\sin u\,\cos v))(0,0) = 0\\
G(\boldsymbol{s})(0,0) &= (\boldsymbol{s}_2\cdot\boldsymbol{s}_2)(0,0) = ((0,1,\sin u\,\cos v)\cdot(0,1,\sin u\,\cos v))(0,0) = 1
\end{aligned}$$

Implementation The functional elements E, F and G of the metric tensor are easily computed in `PLaSM` as the functions `E1`, `F1` and `G1` of Script 5.7.4. According to their defining formulas (5.6), such functions are implemented as the composition of the `innerprod` operator (given in Script 3.2.4) with the appropriate `CONS` of partial derivation operators `DS:`i given in Script 5.7.1.

Script 5.7.4 (First fundamental form)

```
DEF E1 = innerprod ~ [DS:1, DS:1];
DEF F1 = innerprod ~ [DS:1, DS:2];
DEF G1 = innerprod ~ [DS:2, DS:2];

E1:surf:<0,0> ≡ 1.0
F1:surf:<0,0> ≡ 0.0
G1:surf:<0,0> ≡ 1.0
```

5.7.2 Second fundamental form

The tensor of the *second fundamental form* is defined by the matrix:

$$\begin{pmatrix} L & M \\ M & N \end{pmatrix}$$

where the linear forms L, M and N of the tensor are respectively defined as:

$$\begin{aligned} L &= \boldsymbol{n} \cdot \frac{\partial^2}{\partial u^2} \\ M &= \boldsymbol{n} \cdot \frac{\partial^2}{\partial u \partial v} \\ N &= \boldsymbol{n} \cdot \frac{\partial^2}{\partial v^2} \end{aligned}$$

Example 5.7.4
Let us compute the values of the L, M and N forms on the $(0,0)$ corner of the $[0,1]^2$ domain of the $\boldsymbol{s} = (u, v, \sin u \, \sin v)$ surface.

In this case we have

$$\begin{aligned} \boldsymbol{n} &= \boldsymbol{s}_1 \times \boldsymbol{s}_2 \\ &= (1, 0, \cos u \, \sin v) \times (0, 1, \sin u \, \cos v) \\ &= (-\cos u \, \sin v, \sin u \, \cos v, 1) \end{aligned}$$

and

$$\begin{aligned} \boldsymbol{s}_{11} &= \frac{\partial^2 \boldsymbol{s}}{\partial u^2} = (0, 0, -\sin u \, \sin v) \\ \boldsymbol{s}_{12} &= \frac{\partial^2 \boldsymbol{s}}{\partial u \partial v} = (0, 0, \cos u \, \cos v) \\ \boldsymbol{s}_{22} &= \frac{\partial^2 \boldsymbol{s}}{\partial v^2} = (0, 0, -\sin u \, \sin v) \end{aligned}$$

so that

$$\begin{aligned} L(\boldsymbol{s})(0,0) &= (\boldsymbol{n}\cdot\boldsymbol{s}_{11})(0,0) \\ &= ((-\cos u\,\sin v, \sin u\,\cos v, 1)\cdot(0,0,-\sin u\,\sin v))(0,0) = 0 \\ M(\boldsymbol{s})(0,0) &= (\boldsymbol{n}\cdot\boldsymbol{s}_{12})(0,0) \\ &= ((-\cos u\,\sin v, \sin u\,\cos v, 1)\cdot(0,0,\cos u\,\cos v))(0,0) = 1 \\ N(\boldsymbol{s})(0,0) &= (\boldsymbol{n}\cdot\boldsymbol{s}_{22})(0,0) \\ &= ((-\cos u\,\sin v, \sin u\,\cos v, 1)\cdot(0,0,-\sin u\,\sin v))(0,0) = 0 \end{aligned}$$

Implementation In accordance with the implementation of the functional elements of the metric tensor, we give in Script 5.7.5 an implementation of the elements L, M and N of the second fundamental form, respectively denoted as `L2`, `N2` and `M2` in the implementation.

Script 5.7.5 (Second fundamental form)

```
DEF L2 = innerprod ~ [N, DS:1 ~ DS:1];
DEF M2 = innerprod ~ [N, DS:1 ~ DS:2];
DEF N2 = innerprod ~ [N, DS:2 ~ DS:2];

L2:surf:<0,0> ≡ 0.0
M2:surf:<0,0> ≡ 1.0
N2:surf:<0,0> ≡ 0.0
```

A computational check was also executed, by computing $L(\boldsymbol{s})(0,0)$, $M(\boldsymbol{s})(0,0)$ and $N(\boldsymbol{s})(0,0)$, and we found numerically the results already symbolically obtained in the above paragraph.

5.7.3 Gauss curvature

Let consider the normal $\boldsymbol{n}(u,v)$ to a point $\boldsymbol{s}(u,v)$ of a surface $S = \boldsymbol{s}[0,1]^2$ in $\mathbb{E}^3$ and a unit tangent vector $\boldsymbol{v}$ to such a surface in a point $\boldsymbol{s}(u,v)$. Then consider the curve $\boldsymbol{\alpha_v}$ generated by sectioning the surface with the normal plane passing for $\boldsymbol{n}$ and $\boldsymbol{v}$. The curvature of the osculating circle to $\boldsymbol{\alpha}_v$ is called the *normal curvature of* S *at* $\boldsymbol{s}(u,v)$ *in the* $\boldsymbol{v}$ *direction*, and is denoted as $k_n(\boldsymbol{v})$. This function varies between maximum and minimum values, which are reached at two orthogonal directions, called *principal directions*. The maximum and minimum normal curvatures at a point are called *principal curvatures* k_1 and k_2 at that point.

The product of the principal curvatures is called the *Gauss curvature* $K := k_1 k_2$.

The *Gauss curvature field* on a surface $\boldsymbol{s} : [0,1]^2 \to S \subset \mathbb{E}^3$ can be defined as the ratio of determinants of the matrices of the second and first fundamental forms:

$$K(\boldsymbol{s}) : S \to \mathbb{R} : \frac{L(\boldsymbol{s})N(\boldsymbol{s}) - M^2(\boldsymbol{s})}{E(\boldsymbol{s})G(\boldsymbol{s}) - F^2(\boldsymbol{s})}$$

Implementation We already used the symbol `N` to denote the normal field to a surface. Hence in Script 5.7.5, we used the symbols `L2`, `M2` and `N2` to denote the

elements of the second fundamental form. In Script 5.7.6 we give the

$$\texttt{GaussCurvature} : ([0,1]^2 \to \mathbb{E}^3) \to (\mathbb{E}^3 \to \mathbb{R})$$

field operator which, when applied to a surface mapping, returns its curvature field. For the sake of readibility, we would like to write the `GaussCurvature` function using only infix operators, as, e.g., in

```
((L * N) - (M * M)) / ((E * G) - (F * F))
```

but this is not actually possible with the current `PLaSM` implementation, since the algebraic operators would be "raised" too many times. Hence, the standard `FL` prefix and `CONS`ed form must instead be used.

Script 5.7.6 (Gauss curvature)

```
DEF GaussCurvature =
   / ~ [-~[*~[L2,N2], *~[M2,M2]], -~[*~[E1,G1],*~[F1,F1]]];

GaussCurvature:surf:<0,0> ≡ -0.9999999999333331
GaussCurvature:surf:<PI/2,PI/2> ≡ 1.000000165454544
```

Clearly, the given computational examples are affected by approximation and round-off errors, as should be clear by remembering that we are using numeric derivation methods. Once again, the `surf` $= (u, v, \sin u \sin v)$ mapping, used for the two checks above, is the one given in Script 5.7.2.

5.8 Examples

An example of computation and presentation of the Gauss curvature field on a surface, generated by interpolation of two extreme curves with assigned extreme derivative fields, is given in this section. The surface generation by "transfinite" interpolation of curve maps, sometimes called *skinning* in CAD systems, is discussed in Chapter 12.5, but is anticipated here to work out a realistic example.

5.8.1 Color map of the Gauss curvature field

A very interesting application of many concepts studied in this book is discussed here. It allows experimentation with several features of the `PLaSM` language at the same time. In particular:

1. A smooth surface $\boldsymbol{s} : [0,1]^2 \to S$, with $S \subset \mathbb{E}^3$, is generated by using transfinite interpolation methods. For this purpose two boundary curves

$$\boldsymbol{c}_1 \quad : \quad [0,1] \to S : u \mapsto \boldsymbol{s}(u,0) \tag{5.7}$$

$$\boldsymbol{c}_2 \quad : \quad [0,1] \to S : u \mapsto \boldsymbol{s}(u,1), \tag{5.8}$$

   to be interpolated by $\boldsymbol{s}$ are given, and are also assigned two fields of first derivatives along the images of such curves.

2. The field $K : S \to \mathbb{R}$ of Gauss curvature is *sampled* at the vertices of a cell decomposition defined upon the $\boldsymbol{s}[0,1]^2$ image of the surface. The result is represented as a polyhedral approximation, called `CurvatureField`, of the 2D manifold
$$(\mathrm{id}, K)(\boldsymbol{s})[0,1]^2$$
embedded in 4-dimensional space $\mathbb{E}^3 \times \mathbb{R}$.
3. Finally, the 4-th coordinate of vertices of this object is checked for positivity/negativity and used to label each surface vertex with a color triplet in two different ranges of colors, thus transforming the 2-dimensional manifold `CurvatureField` in 4D space into the 2-dimensional manifold `colorGauss` in 6D space.
   (a) At exporting time into a VRML file, the first three coordinates are interpreted as position of points, the last three coordinates are interpreted as *color per vertex* (see for further details Section 10.7).
   (b) The resulting surface, shaded between *cyan* and *white* where the Gauss curvature is positive, and between *blue* and *cyan* where the curvature is negative, is shown in Figure 5.17.

Clearly, such a very simple computational solution to a quite complex scientific visualization problem, was possible because of the dimension-independent `PLaSM` approach to geometric modeling.

Surface generation First of all, we generate a surface mapping $\boldsymbol{s}(u,v)$, with $(u,v) \in [0,1]^2$, by Hermite[1] interpolation of two boundary curves
$$\boldsymbol{s}(u,0) = \mathtt{c1} \quad \text{and} \quad \boldsymbol{s}(u,1) = \mathtt{c2},$$
and two boundary fields of (constant) first derivatives
$$\boldsymbol{s}_2(u,0) = (\underline{1},\underline{1},\underline{1}) \quad \text{and} \quad \boldsymbol{s}_2(u,1) = (\underline{-1},\underline{-1},\underline{-1}).$$

It is not necessary to understand here how the $\boldsymbol{s}$ mapping is generated. The used "transfinite" interpolation is discussed in Section 12.5. Anyway, we have:
$$\boldsymbol{s}(u,v) = \boldsymbol{s}(u,0)\boldsymbol{h}_0(v) + \boldsymbol{s}(u,1)\boldsymbol{h}_1(v) + \boldsymbol{s}_2(u,0)\boldsymbol{h}_2(v) + \boldsymbol{s}_2(u,1)\boldsymbol{h}_3(v),$$
where $(\boldsymbol{h}_0(v), \boldsymbol{h}_1(v), \boldsymbol{h}_2(v), \boldsymbol{h}_3(v))^T$ is the Hermite's basis of cubic polynomials.

Implementation We are interested here to the `Surface` mapping, generated by the `CubicHermite` operator given in Script 12.5.5. Analogously, `c1` and `c2` are two cubic Hermite's curves generated by giving two extreme points and two extreme tangents. Finally, `dom` is a polyhedral complex partitioning the real interval $[0,1]$ into 12 segments, and `CurvatureField` is the resulting polyhedral approximation of the 2D manifold $(\mathrm{id}, K)(\boldsymbol{s})[0,1]^2$ embedded in 4-dimensional space.

---

[1] Discussed in Section 11.2.3 for the standard case of interpolation of points and tangent vectors, and in Section 12.5 for the transfinite interpolation of curves, surfaces and higher dimensional manifolds, together with assigned boundary derivative fields.

Script 5.8.1 (Geometric data)

```
DEF c1 = CubicHermite:S1:<<1,0,0>,<0,1,0>,<0,3,0>,<-3,0,0>>;
DEF c2 = CubicHermite:S1:<<0.5,0,0>,<0,0.5,0>,<0,1,0>,<-1,0,0>>;
DEF Surface = CubicHermite:S2:<c1,c2,<1,1,1>,<-1,-1,-1>>;

DEF dom = interval:<0,1>:12;
DEF CurvatureField =
   MAP: ((CONS ∼ AR ∼ [ID, GaussCurvature]):Surface): (dom * dom);

CurvatureField ≡ A-Polyhedral-Complex{2,4}
```

The transformation from the 2-manifold `CurvatureField` in 4D space, encoding the Gauss curvature as the 4-th coordinate, to the 2-manifold `colorGauss` in 6D space, encoding such curvature as a RGB color triplet, is performed in Script 5.8.2. For this purpose the maximum and minimum values of the 4-th coordinate of `CurvatureField` vertices are stored into `curvmax` and `curvmin`, respectively, and used to map the curvature values to the appropriate triples in the two color ranges used for positive and negative curvatures, respectively.

Script 5.8.2 (Color map of curvature)

```
DEF colorGauss = MAP:(IF:< IsRealPos ∼ s4,
    [s1,s2,s3,/ ∼ [s4,K:(curvmax - curvmin)],K:1,K:1],
    [s1,s2,s3,K:0,/ ∼ [s4,K:(curvmin - curvmax)],K:1] >):CurvatureField
WHERE
   curvmax = MAX:4:CurvatureField,
   curvmin = MIN:4:CurvatureField
END;

VRML:(colorGauss CREASE (PI/2)):'out.wrl';
colorGauss ≡ A-Polyhedral-Complex{2,6}
```

Figure 5.17 Gauss curvature field: (a) surface generated by Hermite's transfinite interpolation of two curves and two (constant) vector fields (b) subset of negative Gauss curvature (c) subset of positive Gauss curvature

Part II

Graphics